Universidade Federal do Rio Grande do Sul
Escola de Engenharia
Departamento de Engenharia Elétrica
Graduação em Engenharia Elétrica

Raissan Chedid

PROTÓTIPO DE UMA BATERIA ELETRÔNICA MUSICAL SEM FIO

Porto Alegre - RS
2015

Raissan Chedid

PROTÓTIPO DE UMA BATERIA ELETRÔNICA MUSICAL SEM FIO

Projeto de Diplomação apresentado ao Departamento de Engenharia Elétrica da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para Graduação em Engenharia Elétrica.

Orientador: Prof. Dr. Marcelo Götz

Porto Alegre - RS

2015

CIP - Catalogação na Publicação

Chedid, Raissan
Bateria Eletrônica sem Fio / Raissan Chedid. -- 2015.
112 f.

Orientador: Marcelo Götz.

Trabalho de conclusão de curso (Graduação) -- Universidade Federal do Rio Grande do Sul, Escola de Engenharia, Curso de Engenharia Elétrica, Porto Alegre, BR-RS, 2015.

1. Sistemas Embarcados. 2. Instrumentação. 3. Comunicação Wireless. 4. Áudio Digital. I. Götz, Marcelo, orient. II. Título.

Elaborada pelo Sistema de Geração Automática de Ficha Catalográfica da UFRGS com os dados fornecidos pelo(a) autor(a).

Raissan Chedid

PROTÓTIPO DE UMA BATERIA ELETRÔNICA MUSICAL SEM FIO

Este projeto foi julgado adequado para fazer jus aos créditos da Disciplina de "Projeto de Diplomação", do Departamento de Engenharia Elétrica e em sua forma final pelo Orientador e pela Banca Examinadora.

Banca examinadora:

Prof. Dr. Marcelo Götz, UFRGS
Doutor pela Universität Paderborn – Paderborn, Germany.

Prof. Dr. Alexandre Balbinot, UFRGS
Doutor pela Universidade Federal do Rio Grande do Sul – Porto Alegre, Brasil.

Prof. Dr. Renato Ventura Bayan Henriques, UFRGS
Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

---

Prof. Dr. Ály Ferreira Flores Filho
Doutor pela University Of Wales College Of Cardiff - UWCC, Gales.

Porto Alegre - RS
2015

RESUMO

Este trabalho contempla o projeto e a implementação de um protótipo experimental de uma bateria eletrônica sem fio. A intenção do projetista na escolha deste trabalho foi de juntar a experiência musical que ele possui com a Engenharia Elétrica. Ele consiste na aquisição de sinais analógicos oriundos de um sensor piezoelétrico condicionado quando a peça do instrumento musical é percutida. O sensor piezoelétrico conseguiu captar os impactos realizados pelo usuário, porém a falta de informações sobre o sensor impossibilitou uma modelagem elétrica e um condicionamento adequado, fornecendo resultados duvidosos. Uma vez adquiridos estes sinais é realizado um pré processamento sobre eles e posteriormente, o valor da amplitude do impacto do usuário é enviado a um módulo de processamento central através de dispositivos de radiofrequência XBee, onde obteve-se resultados aceitáveis visto que o atrasos na transmissão dos sinais não é perceptível para um músico. O módulo central é responsável por realizar o processamento final da informação recebida a fim de reproduzir em um alto falante o som respectivo à peça tocada, além de estabelecer uma comunicação com o computador via protocolo MIDI possibilitando a gravação da execução do usuário. A execução do sons pelo bloco de processamento central apresentou fidelidade ao arquivo de áudio. A gravação via MIDI com o computador não apresentou erros. Os sons da bateria podem ser facilmente alterados, já que estes estão gravados em um cartão de memória SD no formato sem compressão do tipo WAVE.

Palavras-chave: XBee. bateria eletrônica. sistemas embarcados. instrumentação. piezoelétrico.

ABSTRACT

This work contemplates the design and implementation of an experimental prototype of an wireless electronic drums. The purpose of the designer in choosing this work was to join the musical experience he has with Electrical Engineering. It consists in the acquisition of analog signals from a conditioned piezoelectric sensor when the piece of the musical instrument is played. The piezoelectric sensor was able to capture the impact made by the user, but the lack of information about the sensor prevented an electrical modeling and proper conditioning, providing questionable results. After the acquisition of these signals, a pre processing is performed and thereafter the user beat amplitude value is sent to a central processing module through XBee radiofrequency devices, which gave acceptable results since the delays in transmission of the signals are not noticeable to a musician. The central module is responsible for performing the final processing of the information received in order to reproduce in a speaker the sound played drum, and establish communication with the computer via MIDI protocol allowing the recording of execution. The execution of sounds by the central processing unit showed fidelity to the audio file. The recording via MIDI with the computer showed no errors. The sounds of the drums can be easily changed, as they are stored on an SD memory card in WAV without compression.

Keywords: XBee. eletronic drums. embedded systems. intrumentation. piezoelectric.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

EPROM	Memória somente leitura programável e apagável (do inglês “erasable programmable read-only memory”)
MIDI	Interface digital de instrumentos digitais(do inglês *Musical Instrument Digital Interface*)
ASCII	Código padrão americano para o intercâmbio de informação (do inglês *American Standard Code for Information Interchange*). É um código que codifica um conjunto de 128 sinais: 95 sinais gráficos (letras do alfabeto latino, sinais de pontuação e sinais matemáticos) e 33 sinais de controle.
PELE	É uma membrana que cobre as partes superiores e inferiores de tambores de uma bateria acústica ou eletrônica. Geralmente são feitas de plástico ou pele animal na bateria acústica e de borracha ou malha de fibra de vidro na bateria eletrônica
PRATO	No contexto musical e de instrumentos de percussão, é o nome dado às peças construídas a partir de uma liga metálica

SUMÁRIO

1 INTRODUÇÃO

Este trabalho tem como objetivo o projeto e a implementação de uma bateria eletrônica musical sem fio. Utiliza-se a tecnologia de radiofrequência do módulo XBee para permitir a comunicação entre as peças da bateria e o módulo central, assim eliminando a necessidade do uso de cabos.

Cada peça da bateria terá um módulo XBee que será controlado através de um microcontrolador responsável por adquirir o sinal produzido por um transdutor piezoelétrico e controlar os módulos sem fio. O sinal mecânico realizado pela impacto do usuário é transformado em um sinal elétrico e seu valor enviado para o módulo central via radiofrequência.

O equipamento terá dois blocos principais, o bloco de tratamento do transdutor piezoelétrico e o bloco de processamento central. A Figura 1 aponta o diagrama de blocos das peças da bateria.

O bloco de tratamento do transdutor piezoelétrico, o qual possui diagrama apresentado na Figura 1, é responsável pelas seguintes tarefas:

a) adquirir, através de um módulo conversor analógico-digital de um microcontrolador, os sinais gerados por um transdutor piezoelétrico provenientes de impactos realizados pelo usuário;

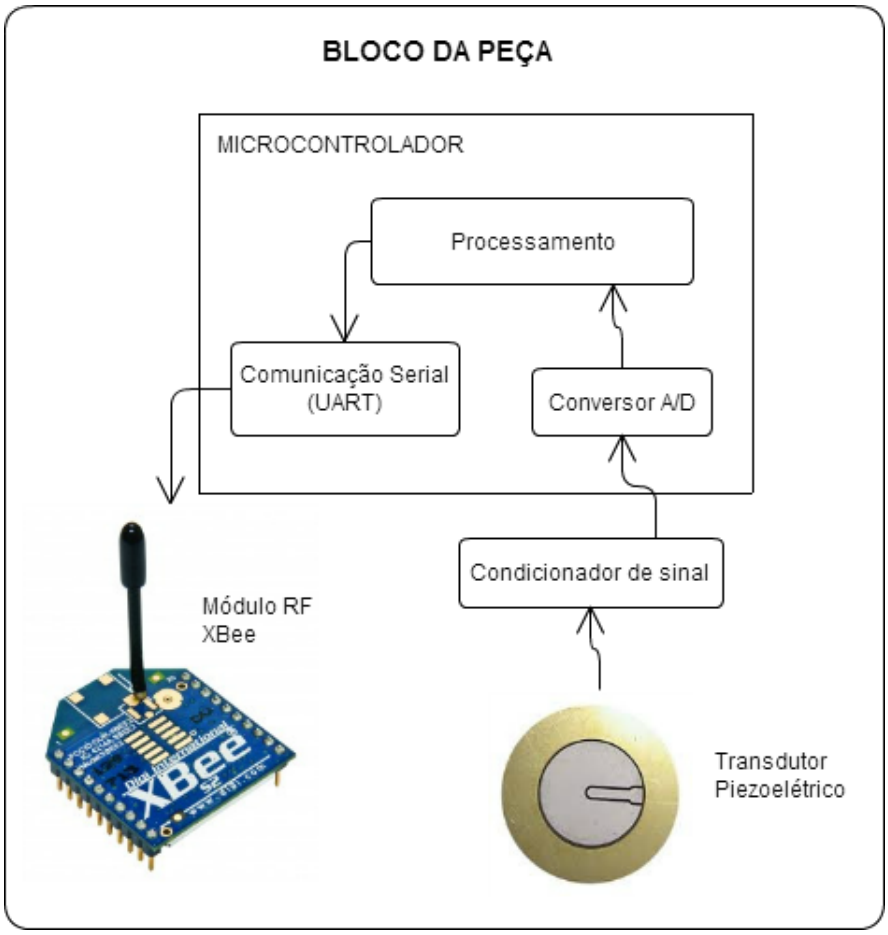
b) realizar o condicionamento da saída em carga fornecida pelo transdutor piezoelétrico a fim de obter o sinal elétrico correspondente;

c) enviar para o bloco de processamento central uma identificação da peça que foi percutida e o valor do sinal adquirido através de dispositivo de comunicação sem fio XBee.

Para este projeto foram implementados três blocos de tratamento dos transdutores piezoelétricos, um para cada peça da bateria. No decorrer do trabalho, os “blocos de tratamento dos transdutores piezoelétricos” serão referidos também como “peças da bateria”, logo o projeto possui três peças para execução, cada uma com o seu respectivo som.

Além das três peças da bateria, foi implementado um bloco de processamento central e seu diagrama de blocos pode ser conferido na Figura 2.

Figura 1: Diagrama de blocos do sensor e da transmissão do sinal de interesse.



Fonte: Elaborado pelo autor.

As funcionalidades do bloco de processamento central são:

a) receber os dados enviados pelas peças da bateria através de um XBee;

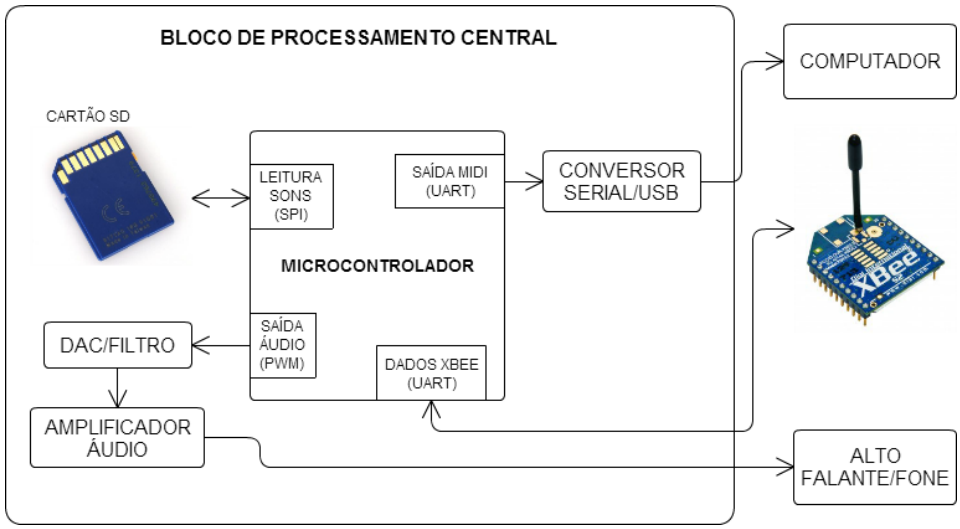
b) identificar qual peça enviou o dado e realizar o processamento deste dado recebido conforme configurações definidas pelo usuário;

c) comunicação com um computador através do protocolo de comunicação entre instrumentos musicais MIDI, o qual está descrito em detalhes na Seção 2.3 e um *software* de gravação musical a fim de possibilitar ao usuário armazenar a execução realizada;

d) leitura dos arquivos de áudio respectivos à cada peça da bateria armazenados em um cartão SD;

e) execução do áudio respectivo à cada peça através da modulação por largura de pulsos (PWM) e a demodulação do mesmo através de um filtro passa baixas recuperando assim o sinal de áudio original, além de um amplificador de áudio;

Figura 2: Diagrama de blocos do processamento central.



Fonte: Elaborado pelo autor.

Uma consideração importante do projeto é o cuidado com o atraso de envio do sinal obtido pelo impacto do usuário para o bloco de processamento central e a execução do som respectivo à peça percutida. No meio musical, esse atraso é chamado de latência. Assim, a latência deve ter um valor que resulte em uma experiência similar à de uma bateria acústica, na qual não existe atraso perceptível entre um toque e a percepção do som. No protótipo implementado, a latência entre o momento do impacto e a execução do som da peça foi definida e ajustada a partir da percepção musical do projetista, assim não se tem o valor quantificado desse tempo de atraso.

O trabalho está estruturado da seguinte maneira: o Capítulo 2 apresenta uma revisão bibliográfica de diversos conceitos e técnicas pertinentes ao projeto. O Capítulo 3 apresenta os métodos utilizados para o projeto e a implementação do equipamento, contendo a descrição de todos os módulos utilizados dos microcontroladores, a maneira como foram adquiridos os sinais das peças da bateria, como foi implementada a comunicação sem fio, a gravação da execução do usuário via protocolo MIDI e a leitura e execução dos sons. Por fim, o Capítulo 4 apresenta os resultados obtidos do protótipo experimental desenvolvido.

2 REVISÃO BIBLIOGRÁFICA

2.1 A Bateria Acústica

A bateria acústica é um instrumento musical que consiste em um conjunto de tambores e pratos, que variam em tamanho e espessura, os quais são percutidos por um só músico que é denominado baterista. Para tocar o instrumento o usuário faz uso de dois ou mais bastões de madeira ou de plástico chamados de baquetas.

Os tambores são formados geralmente por um cilindro de madeira oco com espessura em torno de meio centímetro e profundidade variável definida no projeto, a qual está relacionada diretamente com timbre desejado. Em suas extremidades são utilizadas peles sintéticas onde serão realizadas as impactos onde o som produzido deve-se à vibração destas. Os pratos, por sua vez, são constituídos por metais que quase sempre são latão ou bronze, em diversas proporções. O som produzido pelos pratos, assim como o dos tambores, é devido à sua vibração quando percutidos. Na Figura 3 pode ser visualizada a estrutura padrão de uma bateria acústica.

Figura 3: Estrutura padrão de uma bateria acústica.

Fonte: Adaptado da página sobre bateria acústica no Wikipedia[1].

Baseando-se na Figura 3, o nome dados às peças da bateria e sua descrição são apresentados a seguir. Essas descrições têm como referência a experiência de 15 anos do projetista como músico profissional.

a) o item 1 chama-se prato de condução. Geralmente possui grande diâmetro e espessura a fim de produzir um som parecido com o som de um sino. Leva este nome por possuir um som agudo e seco, próprio para conduzir o ritmo de uma música;

b) o item 2 chama-se surdo. Tambor com grandes dimensões a fim de produzir um som muito grave e sustentado;

c) o item 3 chama-se tom-tons. Tambor com dimensões menores que o surdo, produzindo sons médios;

d) o item 4 chama-se bumbo. Tambor de grande dimensão, produzindo um som grave e seco. Possui um pedal com um batedor, mostrado na Figura 4, possibilitando ser tocado com os pés;

e) o item 5 chama-se caixa. Tambor com profundidade muito pequena onde na pele de baixo possui uma esteira com molas metálicas que vibram junto com a pele, produzindo um som com bastante impacto;

f) o item 6 chama-se chimbal. Consiste em dois pratos virados um para o outro, sendo o prato inferior sustentado em uma base e o prato superior suspenso por uma presilha em um bastão de metal que é ligado à um pedal que, quando pressionado, faz os pratos se chocarem.

2.2 A Bateria Eletrônica

Uma bateria eletrônica possui peças e disposição similares à de uma bateria acústica. Sua principal diferença é que ela não produz som considerável ao ser tocada e os tambores e pratos são substituídos por *pads*[3], que são sustentados por um *rack*[4]. A fim de reproduzir os sons similares ao de uma bateria acústica, os *pads* possuem um transdutor eletromecânico, comumente chamado de *trigger*[5], que transforma a energia mecânica de um impacto em energia elétrica. O sinal elétrico gerado pelo *pad* é enviado através de cabos a um módulo

[1] Disponível em: <https://pt.wikipedia.org/wiki/Bateria_(instrumento_musical)>. Acesso em set. 2015.

[2] Disponível em: <http://www.sweetwater.com/images/items/750/P2000BC-large.jpg>. Acesso em set. 2015.

[3] Peça da bateria eletrônica que simula o tambor de uma bateria acústica. É feita em material elástico, geralmente borracha.

[4] Estrutura que serve para a sustentação dos *pads*, cabos e módulo central da bateria eletrônica

[5] Nome dado ao dispositivo eletromecânico que converte o impacto dado em um *pad* (alguns *pads* são o próprio *trigger*), pele ou aro em um sinal elétrico que será traduzido por um módulo de bateria eletrônica

Figura 4: Pedal de bumbo da fabricante Pearl.

Fonte: Adaptado da página da loja de instrumentos Sweetwater[2].

central que faz o processamento desse sinal, reproduzindo o áudio da respectiva peça percutida.

O módulo pode também possuir uma comunicação com um computador ou com outros dispositivos musicais a fim de enviar a execução do baterista para um programa específico de gravação, para controle de outro equipamento ou simplesmente para servir de espelho para outro módulo de processamento. A comunicação entre dois aparelhos é feita através do protocolo MIDI[6], utilizando uma interface física e lógica para enviar ou receber informações de forma serial.

Na Figura 5 é possível identificar a estrutura padronizada de uma bateria eletrônica da fabricante Yamaha, onde o nome dado às peças destacadas na figura e sua descrição seguem:

a) o item 1 chama-se *pads* dos pratos. Na Figura 5, estes *pads* são feitos de borracha e são fixados ao *rack*;

b) o item 2 é o módulo de processamento. É responsável por todo o processamento dos sinais além de se comunicar com outros dispositivos através do protocolo MIDI. Possui também botões para configuração, *display* para interação com o usuário e saídas de áudio;

c) o item 3 chama-se *pads* dos tambores. Similares aos *pads* dos pratos;

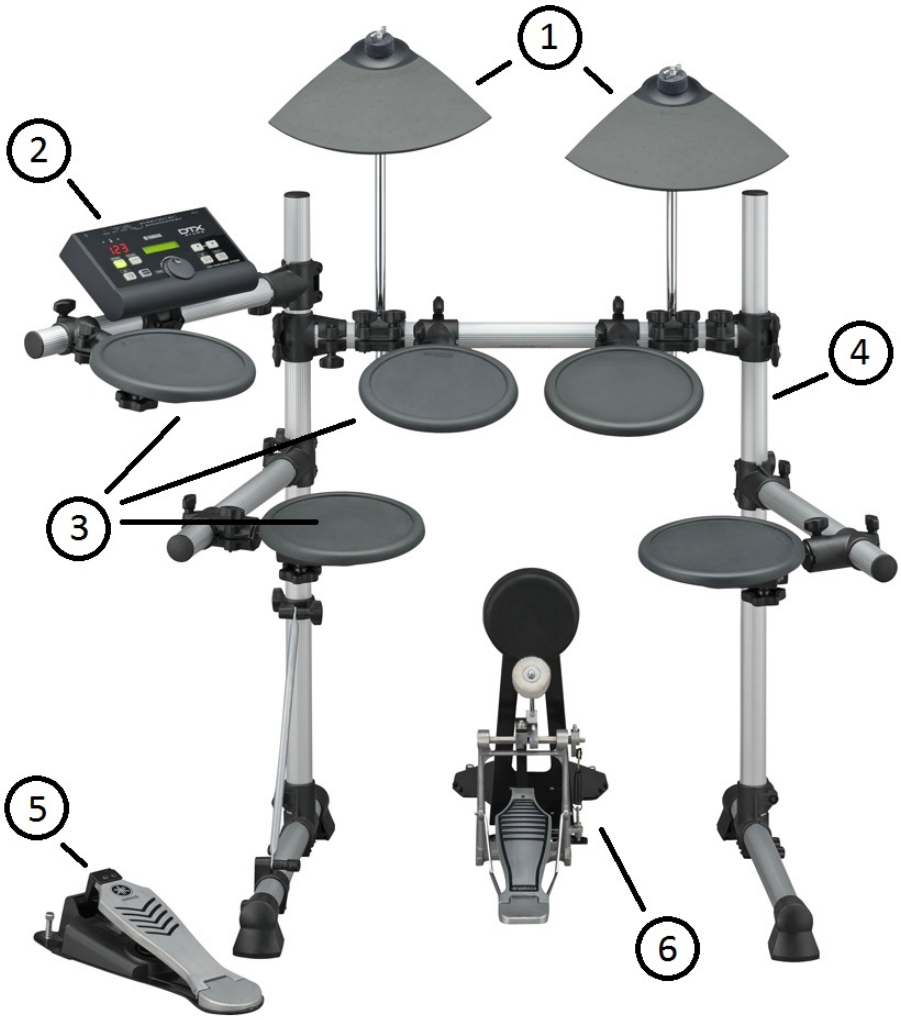
d) o item 4 chama-se *rack*. Sustenta os *pads*, cabos e o módulo, possuindo conexões que possibilitam ajustes de altura e posição das barras;

[6] Protocolo de interface musical detalhado na Seção 2.3

e) o item 5 chama-se pedal do chimbal. Pedal que simula o fechamento dos pratos de chimbal. Pode ser apenas um controle *ON-OFF* ou pode ser dinâmico, possuindo um potenciômetro que varia conforme a pisada, que simula a distância entre os dois pratos;

f) o item 6 chama-se *pad* e pedal de bumbo. *Pads* um pouco diferente dos outros, tendo uma borracha mais dura.

Figura 5: Bateria eletrônica.



Fonte: Adaptado da página da loja de instrumentos Playtech[7].

[7] Disponível em: <http://www.playtech.com.br/bateria-eletronica-yamaha-dtx-520-k-25051.aspx/p>. Acesso em set. 2015.

2.2.1 Histórico

A historia da bateria eletrônica começou a partir da onda dos sintetizadores musicais que se iniciou por volta de 1950. Estes sintetizadores, presentes em sua maioria nos teclados digitais, eram capazes de produzir sons diversos através da manipulação de sinais elétricos analogicamente ou digitalmente. Na década de 1970, Graeme Edge, do grupo Moody Blues, e o professor de eletrônica Brian Groves da Universidade de Sussex desenvolveram o primeiro protótipo de uma bateria eletrônica. Para gerar os sinais elétricos a serem sintetizados era utilizado um ímã permanente que era capaz de se movimentar dentro de uma bobina interligada ao circuito eletrônico principal. Esta estrutura era anexada na parte de trás do *pad* a ser tocado. O sinal gerado era manipulado pelo circuito principal utilizando centenas de transistores, o que tornava este modelo de projeto inviável para ser comercializado. (BADNESS, 1991)

Figura 6: Primeira bateria eletrônica.

Fonte: Adaptado do blog Synthfind[8].

No ano de 1976 foi lançada a primeira bateria eletrônica vendida comercialmente pela empresa *Pollard Industries*. Com o nome de *Syndrum*, o equipamento apresentava um

[8] Disponível em: <http://www.synthfind.com/wp-content/uploads/2011/12/syndrum-quad.jpg>. Acesso em set. 2015.

circuito eletrônico analógico que modificava os sons através de filtros que manipulavam ondas características geradas por um gerador de funções. A peça a ser tocada servia como um gatilho para o gerador de funções, enviando um sinal para este produzir a onda selecionada, passando pelos filtros escolhidos até chegar nos alto falantes. Após o seu lançamento, diversas empresas entraram no mesmo ramo e começaram a produzir seus próprios modelos, como consequência muitas inovações começaram a surgir. O equipamento pode ser conferido na Figura 6. (BADNESS, 1991)

Em 1978 foi lançado pela empresa britânica *Simmons* uma bateria eletrônica, de modelo SDS1, capaz de ter seus sons alterados através da troca da EPROM[9] contida nela. Eram oferecidos diferentes sons para serem adquiridos. Abaixo segue um trecho extraído do manual do fabricante.

> "O som produzido pela SDS1 é uma gravação digital de uma de bateria verdadeira armazenada numa EPROM. A EPROM é ligada à parte frontal da bateria e pode ser facilmente alterada, permitindo que o som do tambor seja alterado. A biblioteca de sons está disponível no seu revendedor Simmons, incluindo todos os tambores e sons eletrônicos populares, ou você pode programar seus próprios sons para uso na SDS1 com o Simmons Sampler e um gravador de EPROM. A SDS1 é dinâmica, quanto mais forte você bate no tambor, mais alto e mais brilhante o som é.". (SIMMONS, 1978)

Na Figura 7, pode-se conferir a SDS1 e o local onde é inserida a EPROM, juntamente com algumas amostras de EPROM com sons diferentes.

Após o lançamento da SDS1 pela *Simmons*, o interesse das empresas em produção de baterias eletrônicas aumentou consideravelmente. Grandes fabricantes tais como Yamaha e Roland tornaram-se líderes no comércio deste ramo.

2.3 *Musical Intrument Digital Interface* (MIDI)

MIDI é um protocolo de comunicação que foi estabelecido a fim de possibilitar a interação de instrumentos digitais entre si e outros dispositivos, como computadores e sequenciadores, o qual permite que instrumentos de fabricantes diferentes possam ser interligados com total compatibilidade. (KERR, 2009)

Um sinal MIDI não contém som, somente instruções que descrevem as notas a serem tocadas, sua intensidade e outras informações. O hardware que recebe o sinal é que executa o som conforme os parâmetros recebidos. Como um sinal MIDI contém somente uma série de instruções, é possível escolher qualquer som a ser executado pelo dispositivo

[9] Descrição disponível na lista de abreviatura e siglas.

[10] Disponível em: <http://www.iimusicshops.com/wordpress/wp-content/uploads/2014/02/simmonsSDS1.jpg>. Acesso em set. 2015.

Figura 7: Bateria eletrônica SDS1 da fabricante *Simmons*.

Fonte: Adaptado do site IIMusicShop[10].

que recebeu este sinal, possibilitando uma ampla gama de possibilidades e controle de execução. (HANSEN, 2005)

2.3.1 Transmissão dos dados

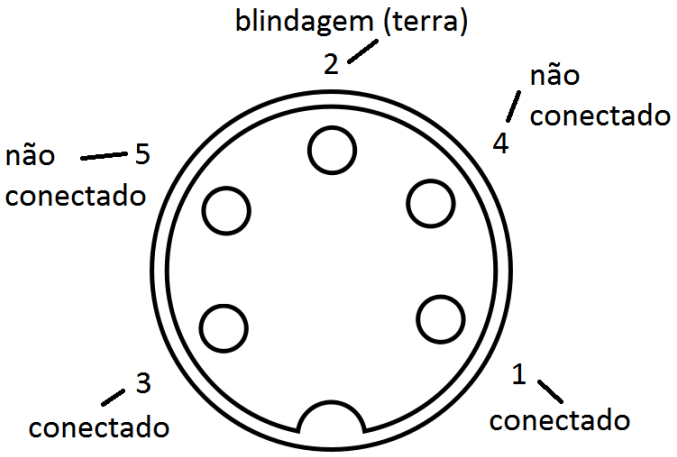
A interface MIDI é basicamente uma interface de transferência de dados em *full duplex*[11]. A transmissão do sinal MIDI é feita através de uma UART de forma serial assíncrona com *baud rate*[12] de 32500 *baud* (+/-1%), com um *start bit* seguidos de 8 bits ou 16 bits de dados e por final um *stop bit*. No total são 10 bits em um período de 320 microssegundos. O *start bit* é identificado com nível lógico zero e o *stop bit* é identificado com nível lógico 1. Os *bytes* são enviados com o bit menos significativo primeiro, da mesma forma que é feita a transmissão de caracteres em ASCII. (COMITTEE, 1995)

A ligação entre interfaces físicas no padrão MIDI é feita através de conectores DIN de 5 pinos. Somente três pinos dos cinco presentes no conector DIN são utilizados. Os dados são enviados através dos pinos 1 e 3, e o pino 2 é ligado ao terra comum do sistema. Os pinos 4 e 5 não são utilizados, como é possível identificar na Figura 8.

[11] Uma comunicação é dita *full duplex* quando temos um dispositivo Transmissor e Receptor, podendo transmitir dados simultaneamente em ambos os sentidos (a transmissão é bidirecional).

[12] O *baud rate* é definido como a taxa de transmissão em *bits* por segundo.

Figura 8: Conexões utilizadas para transferência de dados MIDI.



Fonte: Adaptado de (KERR, 2009)

Segundo (COMITTEE, 1995), é recomendada a utilização de cabos de par trançado e com malha de blindagem a fim de cancelar interferências eletromagnéticas. Além disso, é indicado para os condutores um comprimento máximo de 15 metros e em suas duas extremidades a presença de conectores DIN machos.

Um esquema de ligação está representado na Figura 9, mostrando a diversidade de equipamentos que podem ser interligados via MIDI, tais como sintetizador, módulo de processamento, interface e computador.

2.3.1.1 *IN, OUT* e *THRU*

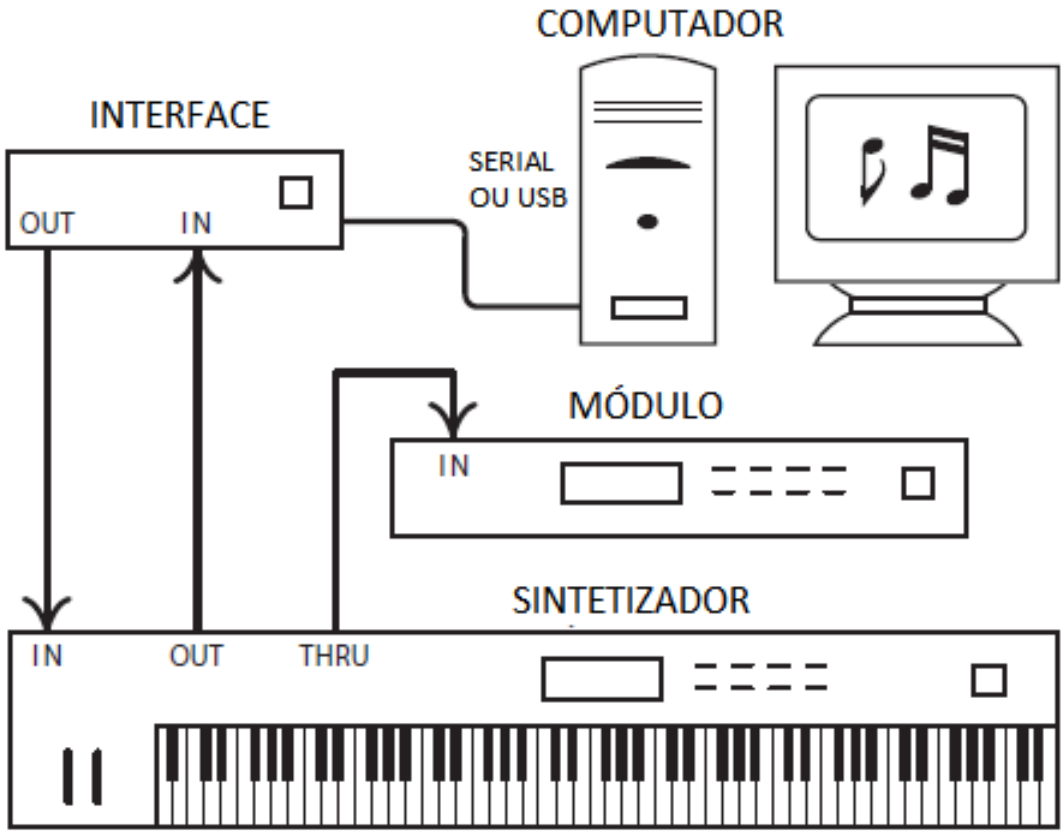
Em um dispositivo padrão que trabalha com MIDI existem a porta de entrada de dados, a porta de saída de dados e uma porta de passagem nos quais são respectivamente chamados de *IN*, *OUT* e *THRU*[13]. As portas *IN* e *OUT* servem simplesmente para receber e enviar dados de outra unidade. A porta *THRU* envia uma cópia exata da informação recebida pela porta *IN* para a porta *OUT* servindo como um retransmissor. Não há nenhuma perda na informação retransmitida, porém há um pequeno atraso gerado no envio da informação. Um esquema de utilização das portas está apresentado na Figura 9. (COMITTEE, 1995)

2.3.2 Estrutura dos dados transmitidos

A comunicação MIDI é feita através de "mensagens" que consistem em um *byte* de identificação (*Status byte*) seguido de um ou dois *bytes* de informações (*Data bytes*),

[13] Palavras da língua inglesa que foram definidas pelo comitê de padrões MIDI do Japão

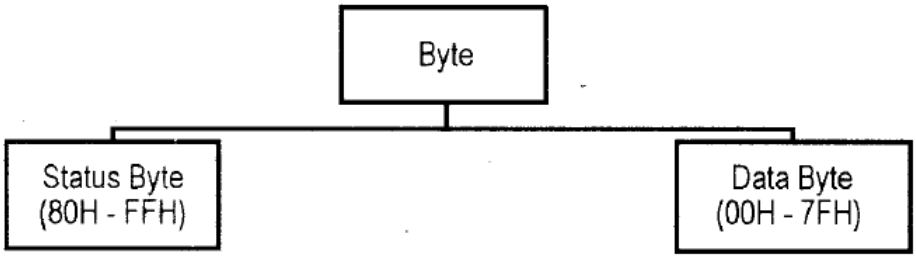
Figura 9: Esquema de ligação entre alguns dispositivos MIDI.



Fonte: Adaptado de (HANSEN, 2005).

com exceção de um tipo especial de mensagem que possui apenas um *byte*. O conjunto desses *bytes*, detalhados na Figura 10, é chamado de mensagem MIDI. O *Status byte* serve para indicar qual o tipo de informação (nota, intensidade, etc.) que está sendo enviada e também para indicar qual o canal da mensagem. Os *Data bytes* são responsáveis por carregar os parâmetros da informação enviada. (KERR, 2009)

Figura 10: Tipos de *bytes* existentes no MIDI.

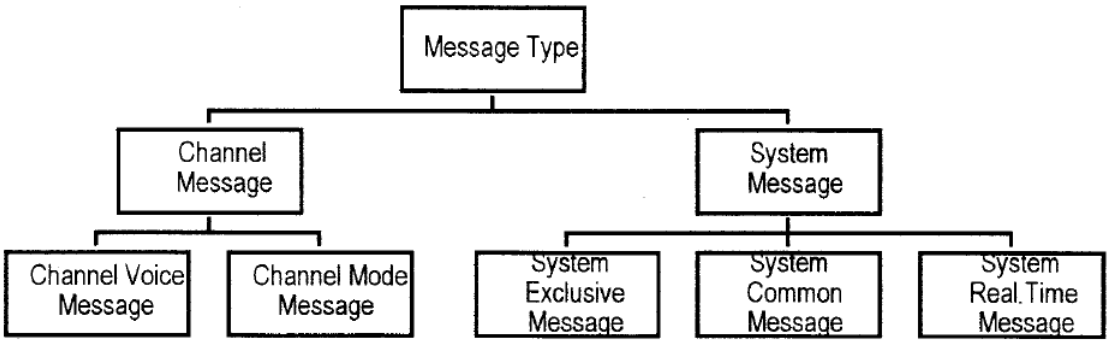


Fonte: Adaptado de (COMITTEE, 1995).

O *Status byte* sempre possui seu *bit* mais significativo com valor lógico "1", enquanto os *Data bytes* sempre possuem seus *bits* mais significativos com valor lógico "0". Isso possibilita ao receptor identificar o tipo de *byte* recebido. Assim, há apenas sete *bits* de dados propriamente ditos, permitindo valores entre 0 e 127. (KERR, 2009)

As mensagens MIDI podem ser enviadas em até 16 canais diferentes, que possibilitam uma variedade de informações da execução musical, e possuem cinco tipos básicos: *"Channel Voice"*, *"Channel Mode"*, *"System Commom"*, *"System Real-Time"* e *"System Exclusive"*. O diagrama presente na Figura 11 mostra a estrutura e a classificação das mensagens transmitidas no protocolo MIDI. (HANSEN, 2005)

Figura 11: Tipos de mensagens MIDI.



Fonte: Adaptado de (COMITTEE, 1995).

2.3.3 *Status Bytes* e *Data Bytes*

Os *bytes* de *status* (*status bytes*) tem a característica de possuírem o seu bit mais significativo com valor lógico 1. *Status bytes* servem para indicar qual o tipo da mensagem, ou seja, o propósito dos *bytes* de dados (*data bytes*) que seguem imediatamente após. Com exceção das mensagens *Real-Time*, novos *Status bytes* sempre comandarão o receptor para adotar um novo estado, mesmo que a última mensagem não tenha sido concluída corretamente. Por sua vez, os *data bytes* possuem o seu bit mais significativo com valor lógico 0 e possuem a informação do parâmetro descrito no *status bytes* enviado antes. (COMITTEE, 1995)

2.4 O Módulo XBee ZB *Series 2*

O módulo XBee ZB é um dispositivo de comunicação sem fio que opera com radiofrequência (RF). Estes módulos foram projetados para operar dentro do protocolo ZigBee e suprir necessidades de baixo custo e consumo de energia em redes de sensores sem fio. Os XBee exigem pouca energia e fornecem uma entrega confiável de dados entre

dispositivos remotos. Os módulos operam dentro da banda de frequência ISM[14] de 2,4 GHz. Os XBee ZB possui uma porta serial para poder se comunicar com qualquer outro dispositivo que possua lógica e tensão compatível. (MAXSTREAM, 2015)

2.4.1 Protocolo ZigBee

ZigBee é o nome dado a um protocolo de comunicação via radiofrequência (RF) que foi criado pela *ZigBee Alliance*. Este protocolo é utilizado pelos módulos XBee. É importante notar a diferença entre XBee e ZigBee. XBee é o nome dado pela empresa Digi ao dispositivo transceptor. ZigBee é uma especificação para a comunicação em uma rede pessoal sem fio (WPAN[15]). Essa especificação tem o objetivo de abranger aplicações com baixo consumo de energia, baixo ciclo de trabalho e dispositivos que exigem baixa taxa de transmissão de dados. O protocolo ZigBee é baseado no padrão IEEE 802.15.4[16], o qual é mantido pelo instituto de Engenheiros Eletricistas e Eletrônicos dos Estados Unidos (IEEE). (INTERNATIONAL, 2015)

2.4.1.1 Formação da rede ZigBee

Uma rede de área pessoal (PAN[17]) ZigBee consiste em um coordenador e um ou mais roteadores e/ou dispositivos finais. Essa rede é criada quando um coordenador seleciona um canal de rede e um PAN ID para iniciar. Uma vez que o coordenador iniciou uma PAN, os roteadores e dispositivos finais são habilitados para entrar na PAN. (MAXSTREAM, 2015)

Quando um roteador ou dispositivo final entra em uma PAN, ele recebe um endereço de rede com 16 bits e pode transmitir ou receber dados de outros dispositivos na PAN. Os roteadores e o coordenador tem a capacidade de permitir outros dispositivos entrarem na PAN e também podem ajudar no envio de dados através da rede a fim de garantir que os dados serão encaminhados corretamente para o dispositivo destinatário. Quando um roteador ou coordenador permite um dispositivo final ingressar na rede, o dispositivo final que ingressou torna-se um filho do roteador ou coordenador que o permitiu entrar. (MAXSTREAM, 2015)

Dispositivos finais podem transmitir ou receber dados, mas não podem rotear os dados de um nó para outro, nem podem permitir que outros dispositivos entrem na PAN. Dispositivos finais devem sempre se comunicar diretamente com o pai. Os dispositivos finais são próprios para serem alimentados por bateria suportam modos de baixa potência. (INTERNATIONAL, 2015)

[14] Abreviatura para *Industrial, Scientific and Medical radio bands*
[15] Abreviatura para *Wireless Personal Area Network*
[16] Padrão que especifica a camada física e o controle de acesso dos dados para redes sem fio pessoais com baixas taxas de transmissão (WPAN)
[17] Abreviatura para *Personal Area Network*

2.5 Transdutor Piezoelétrico

Elementos piezoelétricos são utilizados para construir transdutores para um vasto número de diferentes aplicações. Materiais piezoelétricos apresentam carga elétrica em resposta a um carregamento mecânico, ou vice versa, realizam movimentos mecânicos ao receberem um sinal elétrico. (KARKI, 2000)

O fenômeno conhecido como "piezoeletricidade" foi descoberto nos anos de 1880 pelos irmãos Pierre e Jaques Curie. Em 1881, o efeito piezoelétrico inverso foi reportado. (ABEL; LUIZ; SHIGUE, 2010)

Os cristais piezoelétricos podem gerar tensões elétricas muito altas se submetidos a esforços mecânicos muito intensos. Podem servir para alimentar um sistema elétrico se devidamente tratados. (ADAMOWSKI, 2000)

Um material é considerado piezoelétrico se a aplicação de uma tensão mecânica causa o desenvolvimento de um deslocamento elétrico interno. Este deslocamento se manifesta como uma polarização elétrica interna ou através do aparecimento de cargas elétricas na superfície do material. De todas as classes cristalinas, apenas os representantes com centro de simetria não podem apresentar o efeito. Praticamente todas as outras classes exibem algum efeito piezoelétrico diferente de zero, embora às vezes este efeito seja muito pequeno. Todos os materiais piezoelétricos são transdutores, pois, conseguem converter uma forma de energia em outra (SHACKELFORD, 2008).

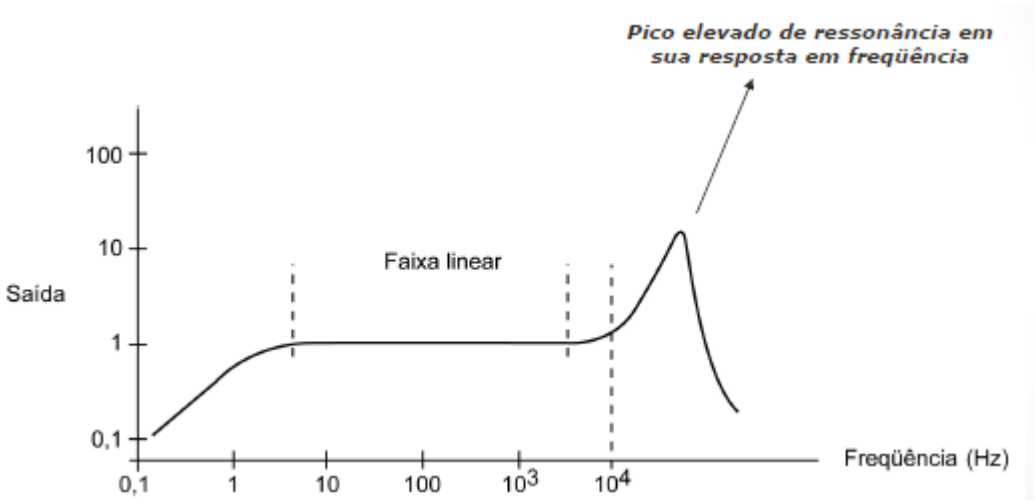
2.5.1 Teoria e Modelagem Elétrica

A teoria básica da piezoeletricidade baseia-se no dipolo elétrico. Ao nível molecular, a estrutura de um material piezoelétrico é tipicamente a de um cristal iônico. Em repouso, os dipolos formados pelos íons positivos e negativos anulam-se mutuamente devido à simetria da estrutura cristalina e um campo elétrico não é observado. Quando submetidos à uma pressão, o cristal é deformado, assim não existindo mais simetria e consequentemente um momento de dipolo, responsável pela formação de um campo elétrico no cristal, surge no material. Deste modo, os materiais geram carga elétrica que é proporcional à pressão aplicada. Se for aplicada periodicamente, uma tensão alternada é observada nos terminais do dispositivo. Sensores piezoelétricos não são adequados para aplicações estáticas ou de corrente contínua pois a carga elétrica produzida decai com o tempo devido à impedância interna do sensor e a impedância de entrada dos circuitos de condicionamento de sinal. No entanto, eles são adequados para aplicações dinâmicas ou com corrente alternada. (KARKI, 2000)

A resposta típica em frequência de um sensor piezoelétrico depende de suas dimensões. Na Figura 12 é possível verificar a resposta típica em frequência de um sensor piezoelétrico. Pode-se perceber que o sensor apresenta inicialmente um comportamento de

um filtro passa-faixa, e um pico de ressonância na frequência de corte superior.

Figura 12: Resposta em frequência típica de um sensor piezoelétrico.



Fonte: Adaptado de (BALBINOT; BRUSAMARELLO, 2011).

Existem diferentes modelos para um transdutor piezoelétrico, desde os mais completos até os mais simples. O modelo simplificado do sensor piezoelétrico é composto por uma fonte de tensão em série com uma capacitância, conforme Figura 13 e não leva em conta as constantes piezoelétricas do cristal. Com este modelo tem-se uma resposta em frequência típica de um filtro passa-alta, onde a frequência de corte é dada pela Equação (2.1), onde a resistência é referente à carga que será alimentada pelo sensor. Este projeto adotou o modelo simplificado como referência devido à falta de informações sobre o transdutor utilizado.

$$f_L = \frac{1}{2\pi RC} \tag{2.1}$$

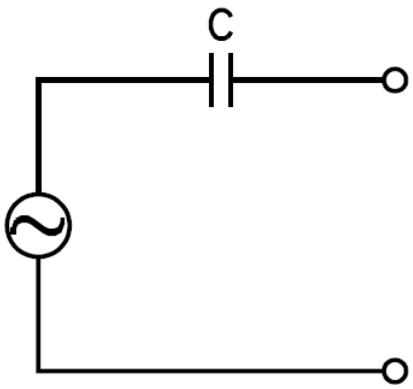
2.6 *Waveform Audio File Format* (WAVE)

O formato de arquivo WAVE é um formato de arquivo desenvolvido pela Microsoft e IBM para armazenamento de áudios. WAVE é a abreviação de *Waveform Audio File Format* (tradução literal: formato de arquivo de ondas de áudio) e é uma variação do método de formatação de arquivos RIFF[18] para o armazenamento de dados em blocos, os chamados *chunks*. Um arquivo RIFF começa com um cabeçalho, seguido por uma sequência de blocos de dados. (MICROSOFT, 1994)

A informação de áudio armazenada no arquivo WAVE nada mais é do que os códigos referentes à tensão amostrada de um áudio analógico, ou seja, armazena um arquivo de

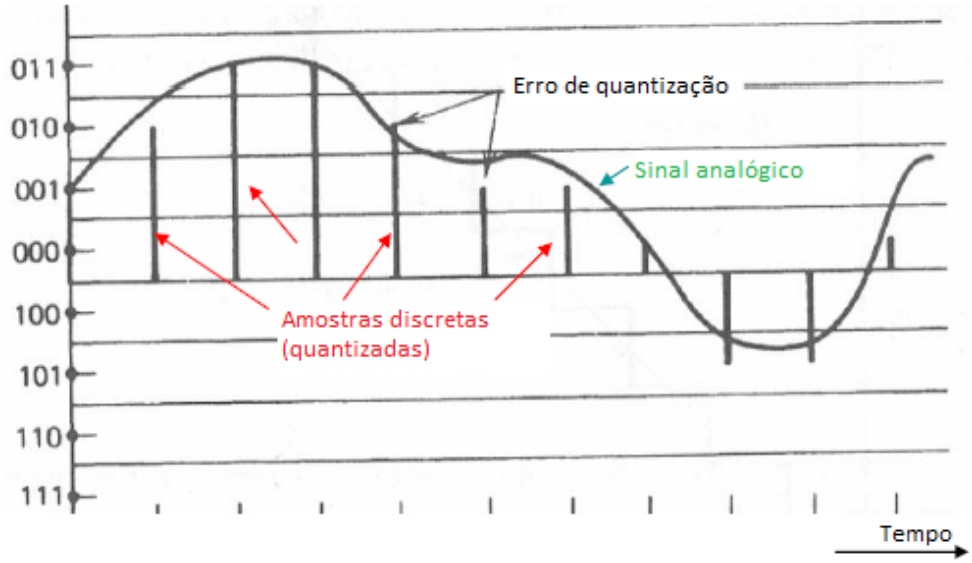
[18] *Resource Interchange File Format*

Figura 13: Modelo simplificado para o transdutor piezoelétrico.



Fonte: Adaptado de (MEASUREMENTSPECIALTIES, 2006).

Figura 14: Exemplo de quantização em uma amostragem.



Fonte: Adaptado de (ALLSYLLABUS.COM, 2015).

áudio na forma de modulação por códigos de pulso (PCM[19]). O PCM é basicamente um método de conversão de um sinal analógico para um sinal digital por meio de amostragem e quantização, isto é, o sinal é amostrado em uma taxa constante e quantificado para o nível de quantização mais próximo. Quantização é o processo de transformar a amplitude do sinal amostrado em um nível discreto baseado em um número finito de valores. Uma demonstração é mostrada na Figura 14. (IBM; MICROSOFT, 1991)

Devido à forma de armazenamento por PCM do áudio analógico, o arquivo WAVE

[19] Do inglês *Pulse Code Modulation*

é um formato sem compressão, ou seja, sem perda de informação. Assim, o WAVE ocupa um espaço muito maior de armazenamento comparado à outros formatos de áudio, porém ele garante a fidelização do sinal de áudio amostrado.

O arquivo WAVE foi escolhido para execução do áudio devido à sua facilidade na leitura dos *bytes* referentes à tensão do sinal de áudio amostrado, pois não é necessário realizar uma decodificação dos dados uma vez que cada *byte* lido corresponde à amplitude do sinal de áudio.

2.6.1 Estrutura do Arquivo WAVE

Os arquivos WAVE têm sua estrutura formada por um cabeçalho (*header*) seguido dos dados do sinal de áudio amostrado. O cabeçalho possui 44 *bytes*, onde informações sobre número de *bits* utilizados na quantização, a taxa de amostragem do sinal de áudio analógico, número de canais e tamanho do arquivo estão presentes. A Figura 15 apresenta a estrutura de um arquivo WAVE. (IBM; MICROSOFT, 1991)

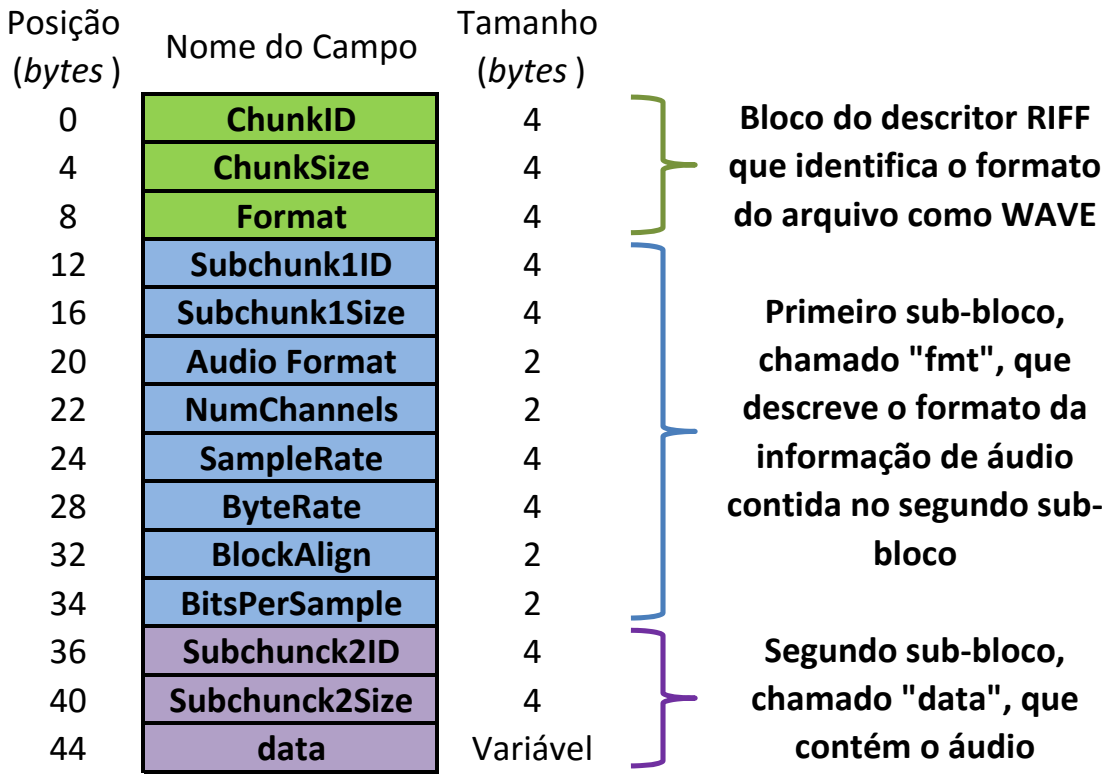
O bloco (*chunk*) inicial do arquivo é um descritor RIFF que tem a função de identificar que o arquivo tem o formato WAVE e também para informar o tamanho do arquivo. Como visto anteriormente, o formato WAVE se baseia no método de formatação RIFF, assim o *chunk* inicial é genérico para qualquer arquivo baseado no RIFF, com a diferença do campo que indica o formato do arquivo. O conteúdo dos campos do *chunk* inicial encontra-se abaixo. (MICROSOFT, 1994)

- ChunkID: campo de 4 *bytes* que contém as letras “RIFF” no formato ASCII (0x52 0x49 0x46 0x46 em hexadecimal). Indica que o arquivo está baseado no método RIFF;
- ChunkSize: campo de 4 *bytes* que contém o valor do tamanho total do arquivo em *bytes* descontando os 8 *bytes* referentes aos campos ChunkID e ChunkSize. Os 4 *bytes* são alocados do *byte* menos significativo para o *byte* mais significativo;
- Format: campo de 4 *bytes* que contém as letras “WAVE” no formato ASCII (0x57 0x41 0x56 0x45 em hexadecimal), indicando que o formato do arquivo é WAVE.

Um arquivo WAVE contém dois sub-blocos que se encontram após o bloco inicial. O primeiro sub-bloco chama-se *fmt* e contém as informações sobre a amostragem e quantização do sinal de áudio e informações pertinentes sobre o áudio. O conteúdo dos campos do primeiro *subchunk* encontra-se abaixo. (MICROSOFT, 1994)

- Subchunk1ID: campo de 4 *bytes* que contém as letras “fmt ” no formato ASCII (0x66 0x6D 0x74 0x20 em hexadecimal), indicando o início do primeiro sub-bloco;

Figura 15: Estrutura de um arquivo WAVE.



Fonte: Elaborado pelo autor.

- Subchunk1Size: campo de 4 *bytes* que contém o tamanho do resto do sub-bloco, a partir desde campo. Para áudios em PCM esse campo tem o valor 16;
- Audio Format: campo de 2 *bytes* que indica se o áudio tem alguma compressão. Se não existe compressão, ou seja, o áudio está modulado por códigos de pulsos, o valor deste campo é 1;
- NumChannels: campo de 2 *bytes* que indica quantos canais o áudio tem;
- SampleRate: campo de 4 *bytes* que contém o valor da taxa de amostragem do sinal de áudio;
- ByteRate: campo de 4 *bits* que indica o número de *bytes* necessários para se ter 1 segundo de áudio. É calculado com a expressão: $(SampleRate * BitsPerSample * Channels)/8$;

- BlockAlign: campo de 2 *bytes* que indica o número de *bytes* necessários para guardar uma amostra do áudio, incluindo todos os canais. É calculado com a expressão: $(NumChannels * BitsPerSample)/8$;
- BitsPerSample: campo de 2 *bytes* que indica quantos bits foram utilizados na quantização do áudio.

O segundo sub-bloco chama-se *data* e contém a quantidade de *bytes* de áudio e os *bytes* amostrados do sinal de áudio. O conteúdo dos campos do segundo *subchunk* encontra-se abaixo. (MICROSOFT, 1994)

- Subchunk2ID: campo de 4 *bytes* que contém as letras "data" no formato ASCII (0x64 0x61 0x74 0x61 em hexadecimal), indicando o início do sub-bloco *data*;
- Subchunk2Size: campo de 4 *bytes* que indica o número de *bytes* do áudio, ou seja, o indica o número de amostras do áudio. Pode ser calculado com a expressão $(NumSamples * NumChannels * BitsPerSample)/8$;
- Data: contém os *bytes* das amostras do áudio, é o áudio propriamente dito. O número de *bytes* deste campo depende da quantidade de amostras do áudio.

Por fim, vale mencionar que as amostras em um arquivo de áudio WAVE PCM multi-canal são intercaladas. Assim, em um arquivo de áudio estéreo (2 canais), uma amostra referente ao canal esquerdo será seguida de uma amostra referente ao canal direito, que por sua vez é seguida de outra amostra do canal esquerdo, assim sucessivamente.

3 METODOLOGIA EXPERIMENTAL

A primeira parte desenvolvida neste projeto foi a comunicação sem fio através dos módulos XBee dos blocos dos sensores com o módulo XBee do bloco de processamento. A comunicação sem fio foi implementada antes pois é a única que não depende das outras funcionalidades. Na sequência, foi desenvolvido o condicionador para o sensor piezoelétrico para poder ser realizada a aquisição e o envio dos sinais do impacto realizado pelo usuário na peça da bateria, a partir de plataformas de madeira que simulam a estrutura de uma peça de bateria. Nesta etapa também foi desenvolvida a rotina para o pré-processamento do sinal proveniente do piezoelétrico.

Após terminada a implementação dos blocos dos sensores, a atenção foi direcionada para o desenvolvimento do bloco de processamento central. Foi então realizado o envio do sinal MIDI para o computador e a gravação de uma execução nas peças da bateria. Posteriormente, implementou-se a leitura e execução dos áudios armazenados em um cartão SD. Todas as etapas do projeto estão detalhadas neste capítulo.

3.1 Módulos XBee ZB

Para a implementação da comunicação sem fio proposta no projeto, foram utilizados quatro módulos XBee ZB. No bloco de processamento foi utilizado um dos módulos como sendo o coordenador da rede, na qual sua função é se comunicar com os outros módulos a fim de receber informações dos impactos na bateria e enviar esses dados para o microcontrolador responsável por processar as informações.

Os outros três módulos XBee ZB foram utilizados nos blocos das peças como roteadores, um módulo para cada peça de bateria. Os módulos roteadores têm a função de enviar o valor da intensidade do impacto do usuário para o XBee ZB do módulo de processamento central, além de enviarem um *byte* de identificação informando qual peça da bateria que está enviando o dado.

Para realizar a configuração, atualização de *firmware* e comunicação para fins de *debug*, foi utilizado o programa gratuito XCTU, fornecido gratuitamente pela fabricante Digi.

3.1.1 Testes dos XBee

O XBee pode operar de dois modos diferentes: o modo AT e o modo API. Os XBee's foram testados nos dois modos a fim de verificar qual deles seria o melhor para a implementação. Os testes se basearam na utilização do módulo XBee sem um microcon-

trolador externo, porém o uso deste microcontrolador se tornou a melhor opção para o projeto, como será mostrado no decorrer desta seção.

As configurações iniciais definidas para teste dos XBee's foram feitas através do *software* XCTU, como seguem:

- Coordenador:
  - *Personal Area Network*(PAN) ID: 1234;
  - Endereço de destino (configurado como *broadcast*) DL: 0xFFFF;
  - *Baud Rate*: 19200.
- Roteadores:
  - PAN ID: 1234;
  - Endereço de destino DH: 0x0013A200 (32 *bits* mais significativos do endereço do coordenador);
  - Endereço de destino DL: 0x4079612A (32 *bits* menos significativos do endereço do coordenador);
  - *Baud Rate*: 19200;
  - RO (*Packetization Timeout*): Zero. Esta configuração define a quantidade de *bytes* do pacote de dados que o XBee irá enviar. Definindo como zero, o XBee transmite o dado assim que ele recebe, evitando a buferização em um pacote.
  - *Sleep Mode*: Sem dormir (*No Sleep*). O módulo XBee dos roteadores foi configurado para não entrar em modo de baixo consumo pois é necessário que os XBee's estejam sempre prontos para enviar os dados.

Algumas dessas configurações sofreram mudanças no decorrer do projeto, tanto para corrigir problemas quanto para implementar melhorias no sistema.

3.1.1.1 Verificação do Modo API

No modo API o XBee possui três funções interessantes. Uma delas é a amostragem periódica das entradas e saídas (*Periodic IO Sampling*), a outra é a amostragem das entradas e saídas a partir de uma identificação de mudança de estado de algum pino (*Change Detection Sampling*) e a última é a amostragem das entradas e saídas a partir de um pedido do coordenador (*Queried Sampling*). Ambas funções incluem a leitura das entradas dos conversores analógico-digitais.

Os testes no modo API foram feitos com o auxílio do *software* gratuito XCTU.

Periodic IO Sampling:

A amostragem periódica consiste em um módulo XBee tirar uma amostra das entradas e saídas, digitais e analógicas, à uma taxa periódica e transmitir essas amostras à um dispositivo remoto. Apenas os dispositivos que estão no modo API podem realizar esta amostragem. Dispositivos configurados no modo AT irão descartar as amostras de dados recebidas.

Foi realizado o teste de envio da amostragem de um pino analógico de um XBee do bloco da peça para o XBee coordenador do bloco de processamento. O período de envio entre as informações foi de 50 milissegundos. O pacote enviado a cada amostragem possuía 22 *bytes*, no formato *frame* característico do modo API. No pacote iam informações como o endereço do XBee que estava enviando, delimitadores de início e fim, quais pinos estavam sendo monitorados, entre outros. Somente um *byte* do pacote era a informação de tensão elétrica convertida. No teste, foram enviados 53 *frames* (totalizando 1166 *bytes* em um período de 2650 milissegundos) ao coordenador e a quantidade de *frames* recebidos foi de 45, demonstrando grande perda de informação em um curto período de tempo. Outra limitação é o período mínimo de amostragem de 50 milissegundos. Para o projeto em questão esta função não se torna adequada pois o equipamento apresentará uma latência muito grande entre o impacto do usuário e a geração do som respectivo à peça, e esta latência será perceptível ao usuário.

Change Detection Sampling:

Os módulos podem ser configurados para transmitir uma amostra de dados sempre que um pino digital sendo monitorado muda de estado. O *datasheet* do XBee não diz se é feita uma amostragem das entradas analógicas também, então foi realizado um teste para constatar se é enviado ou não as amostras dos pinos analógicos.

Com o XBee configurado para adquirir um valor de um pino analógico e monitorar o estado de um pino digital, verificou-se que ao realizar a mudança de estado do pino digital o coordenador recebia somente os valores das entradas digitais, sendo este método inviável para o projeto.

Queried Sampling:

Esta função possibilita ao coordenador solicitar uma amostra do valor das entradas e saídas do XBee.

No *software* XCTU foi criado um *frame* API contendo o comando necessário para solicitar ao roteador a amostragem do valor dos pinos. Ao enviar o *frame* pelo coordenador, obteve-se tempos variados de resposta da amostragem pelo roteador. O tempo de resposta variou conforme a distância entre os módulos, quanto mais longe estava um módulo do outro, mais tempo demorava para o coordenador receber a informação. O tempo mínimo obtido no recebimento do *frame* foi de 37,324 milissegundos, medido no próprio programa

XCTU. Para obter o valor do atraso da informação, foi feita a diferença entre o tempo de envio do *frame* pelo coordenador e o tempo de recebimento do *frame* que continha a amostragem do roteador. Isso foi possível pois o *software* XCTU informa o horário em que foi enviado alguma informação e o tempo que foi recebida alguma informação. Esta implementação foi descartada também por apresentar um atraso maior do que o desejado no projeto.

3.1.1.2 Verificação do Modo AT

O modo AT, por ser um modo "transparente", não possui as possibilidades de implementação que o modo API permite. A palavra transparente se deve ao fato de que o XBee ao receber um dado em sua porta serial, envia imediatamente esse dado por radiofrequência ao dispositivo desejado. Ou seja, no modo AT o XBee serve de "ponte sem fio"para uma comunicação serial.

Este modo torna a implementação da comunicação sem fio muito mais simples e objetiva, uma vez que é enviada o próprio *byte* da informação desejada, e não um *frame* inteiro para um único *byte*.

Nos testes realizados no *software* XCTU não foi possível medir o tempo de atraso entre o envio e o recebimento de dados pois o XCTU não possui esta função para dispositivos operando no modo AT.

Teste de envio periódico:

Neste teste foi utilizada uma função do *software* XCTU onde é possível enviar informações com repetição em intervalos pré definidos.

Em um primeiro momento foi realizado o envio de dois *bytes* (0xD1 e 0xFF) por um roteador para o coordenador, repetindo o envio 250 vezes a uma taxa de 5 milissegundos. O coordenador recebeu os 500 *bytes* corretamente, na sequência correta de envio.

Posteriormente, realizou-se o mesmo tempo descrito acima, porém com três XBee's roteadores enviando dados para o XBee coordenador. Cada roteador enviou um *byte* referente à sua identificação e outro *byte* simulando um valor de intensidade de um impacto do usuário. Um dos roteadores enviou 0xD1 e 0xFF, o segundo enviou 0xD2 e 0xCC e o terceiro enviou 0xD3 e 0xAA. O coordenador recebeu todos os 1500 *bytes* enviados pelos roteadores, porém estavam todos misturados (trecho da sequência de dados recebida, em hexadecimal: D1 FF D2 D3 CC D1 AA D3 FF D2 AA CC), não sendo possível identificar de quem era o valor recebido.

O resultado deste teste eliminou a ideia de monitoramento contínuo dos XBee's, visto que esta metodologia apresentou falhas em ambos os modos API e AT.

3.1.2 Definição do Modo de Trabalho dos XBee

A partir dos testes apresentados na seção anterior decidiu-se utilizar os módulos XBee no modo AT. Além disso, considerou-se para os blocos das peças a implementação de um microcontrolador externo que se comunica com o XBee e também faz a conversão analógica-digital do sinal recebido do condicionador do piezoelétrico. Para o bloco de processamento central, também será utilizado um microcontrolador.

Nos blocos das peças, um módulo conversor analógico-digital presente em um microcontrolador realiza a conversão analógica-digital do piezoelétrico e faz um pré-processamento desse sinal. Após isso, é enviado ao XBee dois *bytes*, um para a identificação da peça outro para o valor do sinal. Decidiu-se utilizar uma conversão analógica-digital de 8 *bits* pois não é necessário ter uma resolução maior para a intensidade de impacto do usuário. Com 8 *bits* tem-se 256 níveis de intensidade para o impacto do usuário e, baseando-se na experiência musical prática e teórica do projetista, com essa quantidade de níveis de intensidade é muito difícil perceber a diferença na amplitude de um nível de intensidade para o próximo. A dificuldade humana em diferenciar muitos níveis de amplitude de impactos realizados em instrumentos é levada em conta na teoria musical, onde existe uma linha de estudo chamada de dinâmica musical que padronizou somente 8 níveis para intensidade de impactos em algum instrumento.

Com esta metodologia, não é necessário utilizar processamento do XBee para a conversão do sinal, deixando toda a capacidade de processamento para a comunicação serial e a radiofrequência. Em outras palavras, o XBee serve de "ponte" *wireless* para o microcontrolador se comunicar com o bloco de processamento central.

Outro ponto da metodologia de transmissão definida, talvez o mais interessante, é que a informação só é enviada quando o usuário realiza um impacto na peça. Assim, quando o microcontrolador identifica um toque no sensor piezoelétrico ele envia os dados pertinentes ao XBee, que por sua vez irá transmitir a informação recebida para o XBee do bloco de processamento central. Essa solução se mostrou mais eficiente comparada ao envio contínuo de dados, visto que é transmitida somente a informação que realmente importa após o impacto realizado pelo usuário. Assim eliminou-se o uso desnecessário do *buffer* de transmissão do XBee evitando problemas no envio, além de reduzir o consumo de energia do bloco assim aumentando a vida útil da bateria utilizada. Neste projeto não foi implementado um método para medir o tempo de atraso entre o momento do impacto e o momento do recebimento do sinal de interesse pelo módulo de processamento central, e para verificar e ajustar esse tempo de atraso utilizou-se a experiência musical do projetista.

3.1.3 Coordenador com *Baud Rate* de 31250 *baud*

Após a implementação do módulo de transmissão do sinal MIDI no bloco de processamento central, percebeu-se a possibilidade de configurar o XBee coordenador para se comunicar com o Arduino Mega 2560 em um *baud rate* de 31250 *baud*. Com essa mudança na taxa de comunicação do XBee (primeiramente sendo 19200 *baud*) foi possível a eliminação de uma USART do Arduino Mega 2560, que estava sendo utilizada somente para o envio da informação MIDI para o computador. Alterando o *baud rate* do XBee para 31250 *baud*, na qual é o *baud rate* padrão para a comunicação MIDI (ver Subseção 2.3.1), utilizou-se somente um módulo de comunicação serial do ATmega 2560. A configuração do *baud rate* do XBee foi feita através de comandos AT enviados pelo terminal serial do *software* XCTU, apresentados no Apêndice A. Vale lembrar que o Arduino só recebe informação do XBee e o módulo USART0 utilizado do ATmega 2560 é *full-duplex*, assim a transmissão MIDI para o computador pode ser feita no mesmo módulo serial.

3.2 Microcontroladores

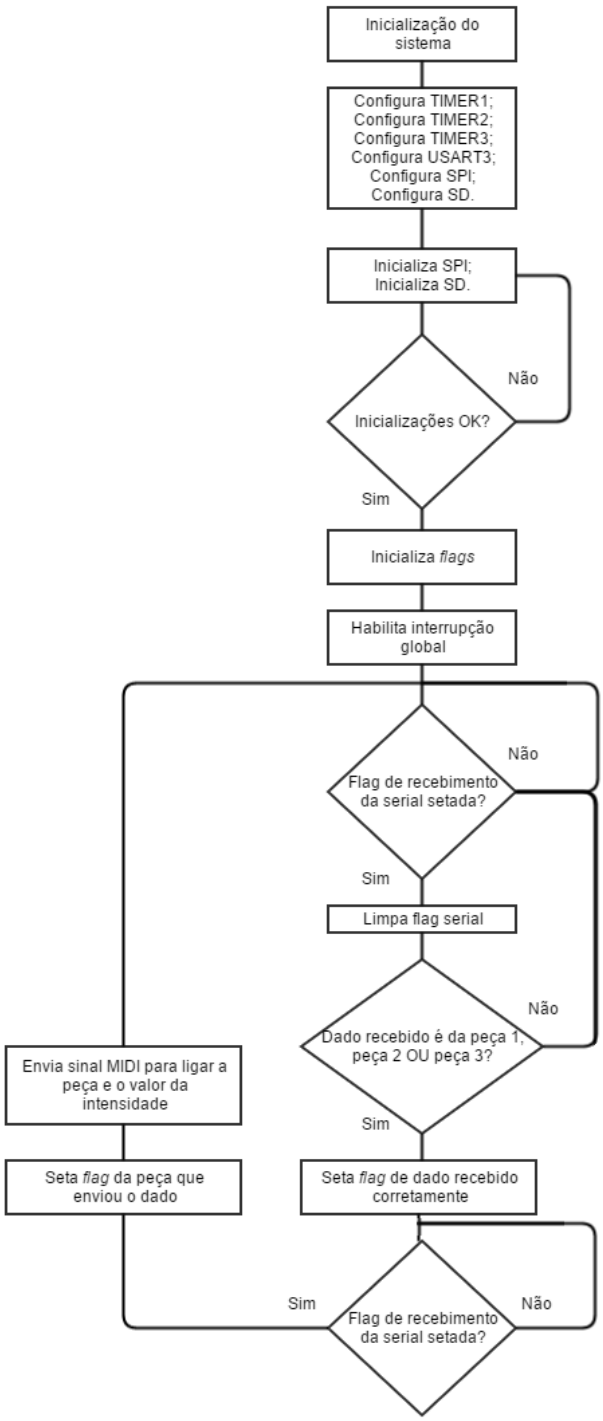
Foram utilizados microcontroladores em todos os blocos. Nos blocos das peças é utilizado um PIC18F2550 que realiza, através de um módulo de conversão analógico-digital interno, a conversão do valor de tensão obtido na saída do condicionamento do piezoelétrico para um valor digital. O microcontrolador também realiza um pré-processamento do sinal adquirido e se comunica serialmente com o XBee roteador. Já para o bloco de processamento central, foi utilizado um o *kit* de desenvolvimento Arduino Mega 2560, no qual possui o microcontrolador ATmega 2560. Este microcontrolador é responsável pela comunicação com o XBee coordenador, pelo processamento dos dados recebidos dos blocos das peças, pela comunicação com o computador via protocolo MIDI, pela leitura dos sons no formato WAVE armazenados no cartão SD e pela execução do áudio de cada peça através de PWM.

3.2.1 Arduino Mega 2560

O Arduino Mega 2560 é uma placa de desenvolvimento baseada no microcontrolador de 8 *bits* e arquitetura RISC avançada ATmega2560 e foi utilizada no bloco de processamento central. Ela possui 54 pinos digitais de entrada/saída (15 destes pinos podem ser utilizados como saídas PWM), 16 entradas analógicas, 4 UARTs (portas seriais) e comunicação SPI. A placa trabalha com uma frequência de 16 MHz e possui o microcontrolador ATmega16U2 que é responsável pela conexão USB com o computador, além de ter todos os componentes necessários para o funcionamento dos microcontroladores. A placa foi escolhida por fornecer todas as funcionalidades necessárias para o desenvolvimento do bloco de processamento central.

O fluxograma de execução das tarefas pelo ATmega2560 está apresentado na Figura 16.

Figura 16: Fluxograma de execução das tarefas do ATmega2560.



Fonte: Elaborado pelo autor.

3.2.1.1 IDE Atmel Studio 6.2

Atmel Studio é uma plataforma de desenvolvimento gratuita disponibilizada pela fabricante Atmel para desenvolver projetos com seus microcontroladores. Esta ferramenta possibilita criar códigos em C/C++ ou Assembly e depurar o programa tanto com ferramentas externas como em um simulador integrado. O ambiente de desenvolvimento integrado (IDE) Atmel Studio utiliza o compilador de código aberto GCC C/C++. A IDE é baseada em outra IDE, o Visual Studio da fabricante Microsoft.

Inicialmente, todo o desenvolvimento dos códigos para o Arduino Mega 2560 foi realizado na IDE Atmel Studio 6.2, utilizando como referência as bibliotecas oferecidas pelo próprio compilador GCC AVR, que podem ser consultadas em <http://www.nongnu.org/avr-libc/>. Porém, para utilizar o cartão SD foi usado a IDE do próprio Arduino, como está explicado na Subseção 3.3.2.1.

3.2.1.2 Code Wizard AVR

Para ajudar no desenvolvimento do código foi utilizada uma ferramenta que realiza geração automática das rotinas de inicialização do processador e de configuração dos registradores de periféricos (SFR) que é o *software* Code Wizard AVR. Este *software* é uma IDE específica para o desenvolvimento de microcontroladores de 8 *bits* da Atmel e possui um gerador de códigos automático para a família de microcontroladores AVR e XMEGA. Através da interface gráfica do *software*, define-se a configuração desejada para o periférico e a ferramenta gera o código de inicialização. Embora esta funcionalidade faça parte de uma IDE e um compilador específico, pode ser utilizada independentemente. A ferramenta também instala um *plug-in* na IDE Atmel Studio que permite o *download* do *firmware* gerado em placas Arduino, facilitando assim seu uso. No decorrer de todo o projeto, foi utilizada uma versão de avaliação gratuita disponibilizada no endereço virtual do fabricante.

3.2.1.3 Execução do Áudio

Uma breve descrição dos módulos utilizados para a execução do áudio encontra-se nesta seção. Para uma descrição detalhada sobre todo o processo necessário para executar o áudio consulte a Seção 3.3.

Para realizar a execução de um arquivo de áudio WAVE gravado em um cartão SD foram utilizados os seguintes periféricos:

- TIMER3: Foi utilizado para gerar interrupções na frequência da taxa de amostragem do arquivo de áudio (16kHz). A cada interrupção, uma rotina é executada para atualizar o valor do *duty cycle* do PWM do TIMER2 com o valor do *byte* lido do arquivo WAVE;

- TIMER2: Foi utilizado para gerar o sinal PWM referente ao áudio lido do cartão SD. A frequência do PWM foi definida como sendo 62,5kHz.;
- SPI: Este módulo foi utilizado para se comunicar com o cartão SD a fim de ler o arquivo WAVE contido no cartão.

3.2.1.4 USART3

O módulo USART do ATmega 2560 é bastante versátil possuindo funcionalidades para a maioria das necessidades de comunicação serial. Possui operação em *full duplex* com registradores para transmissão e recepção independentes entre si, além de disponibilizar interrupções de transmissão terminada, recepção terminada e quando o registrador dos dados de transmissão está vazio. Possui também um gerador de *baud rate* de alta resolução, permitindo a redução de erros relacionados à frequência de comunicação. Basicamente, este módulo USART é dividido em três partes principais: geração do *clock*, transmissão e recepção.

O bloco responsável pela geração do *baud rate* tem seu diagrama de blocos mostrado na Figura 17, onde **txclk** é o *clock* de transmissão, **rxclk** é a base do clock de recepção e **fosc** é a frequência de oscilação do sistema. O *baud rate* é gerado a partir de um valor de 12 *bits* carregado em dois registradores chamados UBRRH e UBRRL[1] em conjunto com um contador decrescente, funcionando a partir do *clock* do sistema. O valor de UBRR é carregado no contador cada vez que este chega ao valor zero ou quando algum valor é escrito em UBRR. No modo de operação assíncrono normal, o *baud rate* é calculado conforme a Equação (3.1) e o valor que deve ser carregado é calculado a partir da Equação (3.2).

$$BAUDRATE = \frac{fosc}{16(UBRRn + 1)} \tag{3.1}$$

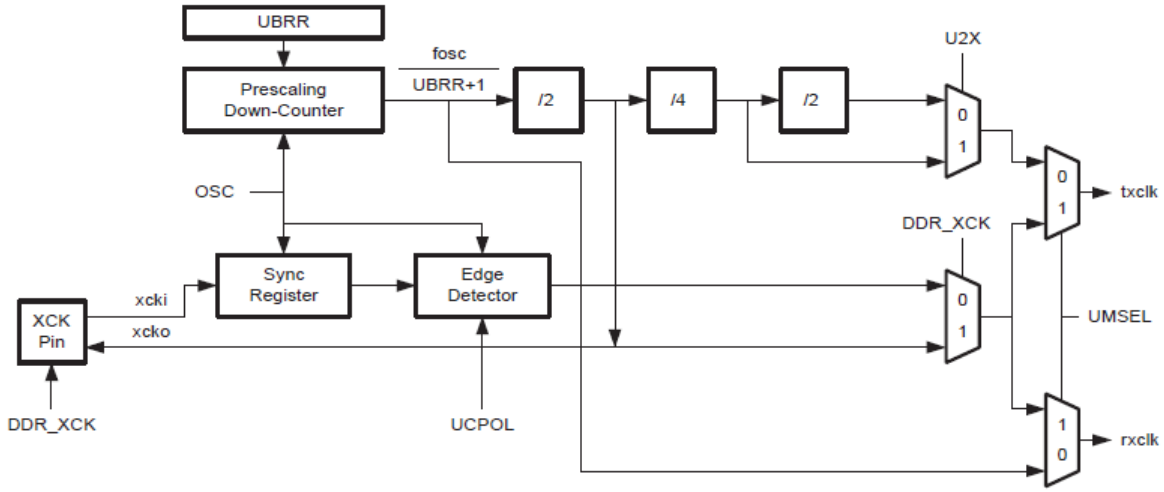
$$UBRR = \frac{fosc}{16BAUD} - 1 \tag{3.2}$$

O *baud rate* da USART foi definido como 31250 *baud*, necessário para a comunicação em MIDI. Como a frequência de oscilação $fosc$ do sistema é 16 MHz, utilizando a Equação (3.2) obtém-se o valor decimal 31 que deve ser carregado no registrador UBRR, apresentando 0% de erro. Utilizando o valor 31 para UBRR na Equação (3.1) obtém-se exatamente 31250 *baud*. O XBee que se comunica com o ATmega2560 também foi configurado com 31250 *baud*, conforme Subseção 3.1.3.

Os dados são enviados através de *frames* conforme Figura 18, definidos com um *start bit*, seguidos de 8 *bits* de dados e por último um *stop bit*. O *start bit* (St) e o *stop*

[1] *Usart Baud Rate Register*

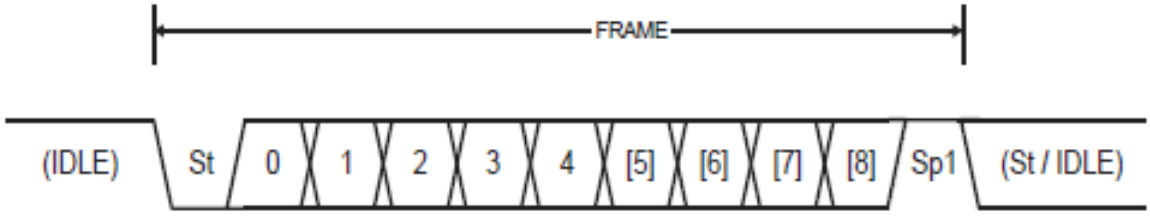
Figura 17: Diagrama de blocos da lógica de geração do *clock* do módulo USART.



Fonte: Adaptado de (ATMEL, 2014).

bit (Sp1) servem para sincronizar o dispositivo que receberá a informação, indicando o início e o fim do *frame*, respectivamente. O *start bit* é definido como padrão sempre com nível lógico 0, sendo o contrário do *stop bit*. Os dados são enviados do *bit* menos significativo para o *bit* mais significativo. Se nenhuma transmissão está acontecendo, a linha de comunicação fica inativa (*IDLE*) com nível lógico 1.

Figura 18: Formato de envio dos dados pela USART.



Fonte: Adaptado de (ATMEL, 2014).

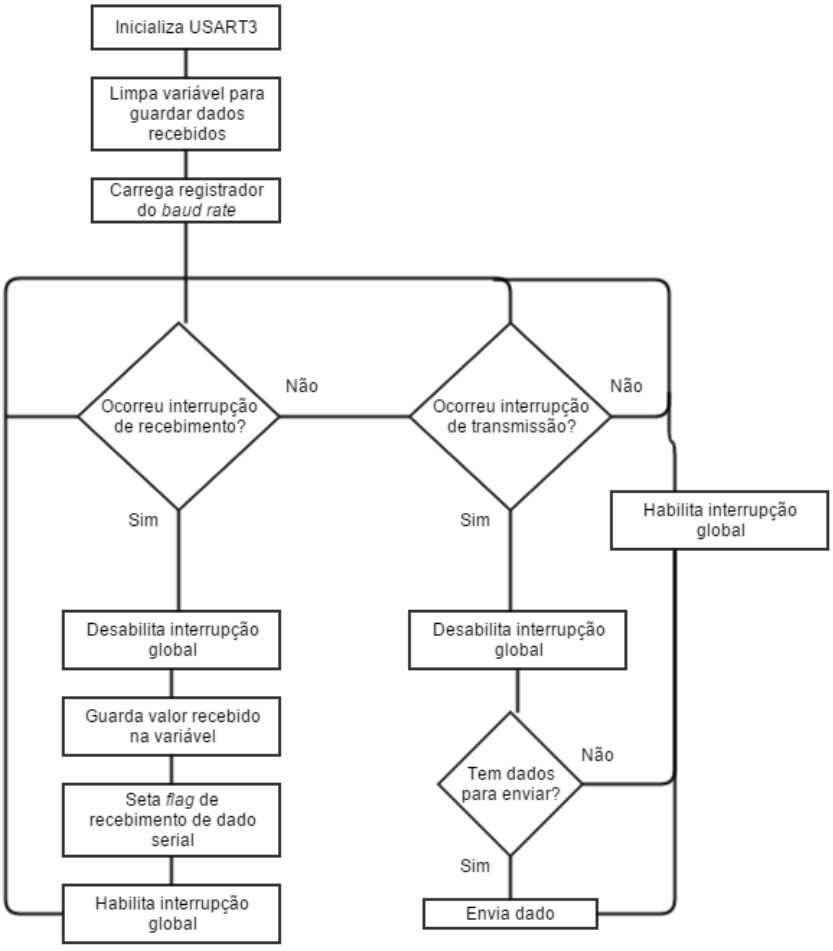
A parte transmissora possui um único *buffer* para escrita permitindo uma transferência contínua de dados sem nenhum atraso entre os *frames*. Uma vez configurada como desejado, uma transmissão de dados ocorre quando algum dado é carregado no *buffer* de transmissão a partir da escrita no registrador UDR e quando o *Shift Register* está pronto para enviar um *frame*. O *Shift Register* é carregado com novos dados imediatamente depois do último *stop bit* do *frame* anterior. Foi utilizado a interrupção de transmissão completa (*Transmit Complete Interrupt*) para tratar o envio dos dados.

Já a parte de recepção dos dados é a parte mais complexa do módulo USART do ATmega 2560 devido às suas unidades de recuperação do *clock* e recuperação dos dados,

que servem para receber dados de uma comunicação assíncrona. A recepção possui um *buffer* de dois níveis FIFO[2], o qual compartilha o mesmo endereço de memória (registrador UDRn) do *buffer* de transmissão. O receptor inicia a recepção dos dados quando detecta um *start bit* válido. Os *bits* de dados seguintes são amostrados na taxa do *baud rate* e deslocados para o *buffer* de recepção. Para tratar a recepção de dados foi utilizado a interrupção de recepção completa (*Receive Complete Interrupt*).

O fluxograma de execução das tarefas do módulo USART3 está apresentado na Figura 19.

Figura 19: Fluxograma de execução das tarefas do módulo USART3.



Fonte: Elaborado pelo autor.

[2] *First In First Out*

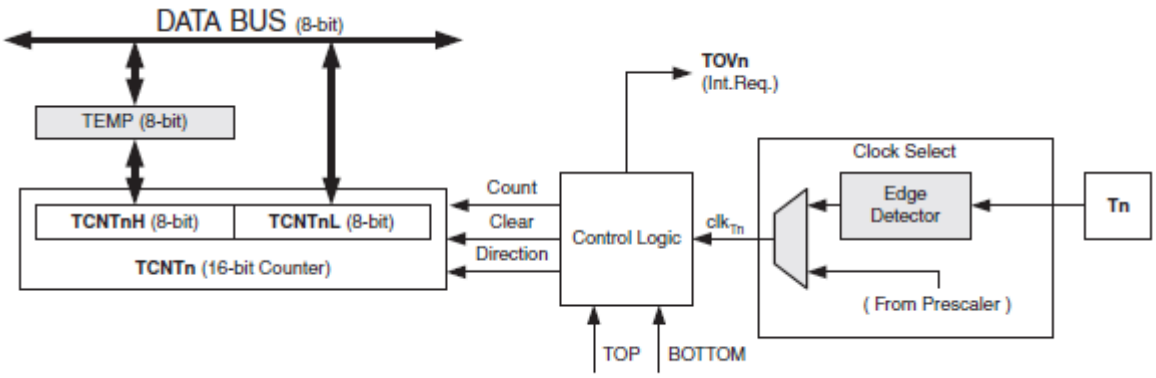
3.2.1.5 TIMER1

O módulo TIMER1 do ATmega2560 é um módulo temporizador de 16 bits, sendo uma ótima solução para realizar temporizações precisas em programas. A temporização é realizada através de um contador de 16 bits e uma interrupção de sistema é realizada quando o contador atinge o valor de 16 bits configurado. O módulo do TIMER1 possui diversas outras funcionalidades além da temporização com interrupção, porém essas funções não serão descritas neste documento.

A função do módulo TIMER1 é proporcionar uma temporização precisa para o sistema poder executar funções e verificações periódicas, como contadores para verificação se alguma peça foi tocada.

Para realizar uma temporização, o TIMER1 possui uma unidade contadora que é responsável pela comparação do valor do registrador de 16 bits TCNT1 configurado pelo usuário com o valor da contagem atual, gerando uma interrupção quando os valores se igualam. O registrador TCNT1 é a junção dos registradores de 8 bits TCNT1H e TCNT1L, os quais guardam os bits mais significativos e os menos significativos, respectivamente. Na Figura 20 é possível visualizar o diagrama de blocos da unidade contadora do TIMER1, onde **Count** é o sinal para incrementar ou decrementar o valor de TCNT1, **Direction** é o sinal de controle que seleciona entre incrementar ou decrementar, **Clear** zera o registrador TCNT1, clk_{T1} é o clock do TIMER1, **T1** é o valor do *prescaler* do TIMER1 e **TOP** é o sinalizador do *overflow* do registrador TCNT1.

Figura 20: Diagrama de blocos da unidade contadora do TIMER1.



Fonte: Adaptado de (ATMEL, 2014).

Para realizar uma leitura ou escrita no contador de 16 bits com apenas um ciclo de *clock* através do barramento de dados de 8 bits, o registrador TNCT1H só pode ser acessado indiretamente pela unidade central de processamento (CPU). Quando é necessário um acesso à TCNT1H, a CPU acessa um registrador temporário para o *byte* mais significativo

(TEMP). O valor do registrador temporário é atualizado com o valor de TCNT1H quando o registrador TCNT1L é lido, e TCNT1H é atualizado com o valor do registrador temporário quando é realizada uma escrita em TCNT1L.

O contador pode ser configurado para ser zerado, incrementado ou decrementado a cada clock. O clock clk_{T1} pode ser gerado por uma fonte externa ou interna. Se a interrupção estiver habilitada, a *flag* TOV1 (*Timer/Counter Overflow Flag*) é setada para gerar uma interrupção no sistema.

No modo de operação normal, o qual foi utilizado no projeto, o contador é incrementado a cada ciclo de clock do TIMER1 do valor zero até o valor definido no registrador TCNT1, sendo gerada uma interrupção quando o contador chega no valor de TCNT1.

O clock do TIMER1 foi configurado sem *prescaler*, assim possui mesmo valor do clock do sistema, 16MHz. A temporização definida para realizar algumas verificações no sistema foi de 1ms. Como a frequência do TIMER1 é de 16MHz, a cada 62,5ns o contador é incrementado. Então, para termos uma interrupção a cada 1ms, o TIMER1 deve contar 16000 pulsos de clock e gerar um interrupção ($1ms/62,5ns = 16000$). Como o TIMER1 gera uma interrupção quando o valor do contador passa de 0xFFFF para 0x0000, a cada interrupção gerada é preciso reinicializar o valor do registrador TCNT1 para que o contador conte 16000 vezes até chegar à 0xFFFF. O valor a ser carregado em TCNT1 a cada interrupção é encontrado fazendo-se $65536 - 16000 = 49536$ (0xFFFF - 0x3E80 = 0xC180). Assim, a cada interrupção do TIMER1, o valor do registrador TCNT1 é recarregado com o valor **0x3180**.

O fluxograma de execução das tarefas do módulo TIMER1 está apresentado na Figura 21.

Figura 21: Fluxograma de execução das tarefas do módulo TIMER1.

Fonte: Elaborado pelo autor.

3.2.2 PIC 18F2550

O PIC 18F2550 é um microcontrolador fabricado pela Microchip e possui diversas funcionalidades, das quais serão utilizadas somente os conversores analógicos-digitais e a comunicação serial. O microcontrolador foi escolhido por disponibilidade e por ter as funções desejadas para o projeto, além de ser da categoria *nanoWatt*[3] dos microcontroladores da família PIC18. O *chip* foi utilizado com uma alimentação de 3.3V por ser a mesma tensão de trabalho dos módulos XBee. A frequência de oscilação escolhida foi de 12MHz pois, segundo (MICROCHIP, 2006) 12MHz é uma frequência estável de trabalho para microcontroladores PIC com tensão elétrica de alimentação de 3.3V.

O PIC 18F2550 foi utilizado em duas etapas. A primeira etapa foi na validação do transdutor piezoelétrico, onde o microcontrolador foi utilizado somente para adquirir dados em uma taxa de amostragem de 5kHz durante 5 segundos e enviá-los pela porta serial a fim de armazenar os dados no computador para análise posterior. Na segunda etapa, o microcontrolador foi utilizado para realizar um pré-processamento do sinal adquirido do piezoelétrico e enviar serialmente o valor da intensidade do impacto do usuário para o módulo XBee.

O módulo USART do microcontrolador foi utilizado para comunicação com o XBee e foi configurada uma taxa de transmissão de dados de 19200 *baud*. O conversor A/D foi utilizado para adquirir os sinais provenientes do condicionador do sensor piezoelétrico. A frequência de amostragem foi definida como sendo de 5kHz.

3.3 Execução do Áudio das Peças

Uma funcionalidade implementada no projeto foi a possibilidade de executar um áudio respectivo à cada peça. O som é executado quando o usuário realiza um impacto em uma das três peças.

O áudio é executado através de um módulo PWM do ATmega2560 que possui uma frequência de aproximadamente 4 vezes a frequência de amostragem do áudio. O valor do *duty cycle* do PWM é controlado através de um módulo temporizador, também do ATmega2560, cuja frequência com que é atualizado o *duty cycle* é exatamente a mesma frequência de amostragem do áudio.

3.3.1 Arquivos WAVE

Os arquivos WAVE utilizados na reprodução dos sons respectivos à cada peça foram adquiridos gratuitamente no sítio virtual SampleSwap[4], que é um repositório de arquivos

[3] Tecnologia criada pela fabricante Microchip que visa otimizar seus *chips* para o mínimo consumo de energia.

[4] Disponível em <http://sampleswap.org/index.php>. Acesso em nov. 2015.

de áudio gratuitos para utilização. A fim de simular o som de uma bateria acústica, foram adquiridos áudios para o bumbo, a caixa e o prato de ataque.

Como as faixas adquiridas eram estéreo (2 canais), foi preciso transformar a faixa para mono (1 canal) através de um *software* de edição de áudio. Utilizou-se o *software* Audacity, que é um programa gratuito que permite editar, gravar, importar e exportar diversos formatos diferentes de arquivos de áudio[5].

Após transformar a faixa para *mono*, foi preciso alterar a sua taxa de amostragem. Inicialmente, a taxa do arquivo era de 44,1kHz. Como o protótipo foi projetado para executar somente áudios com taxa de amostragem de 16kHz, foi preciso realizar uma conversão da taxa de amostragem através do *software* Audacity. Depois das alterações feitas nos áudios os arquivos ficaram com tamanho e duração como indicado abaixo.

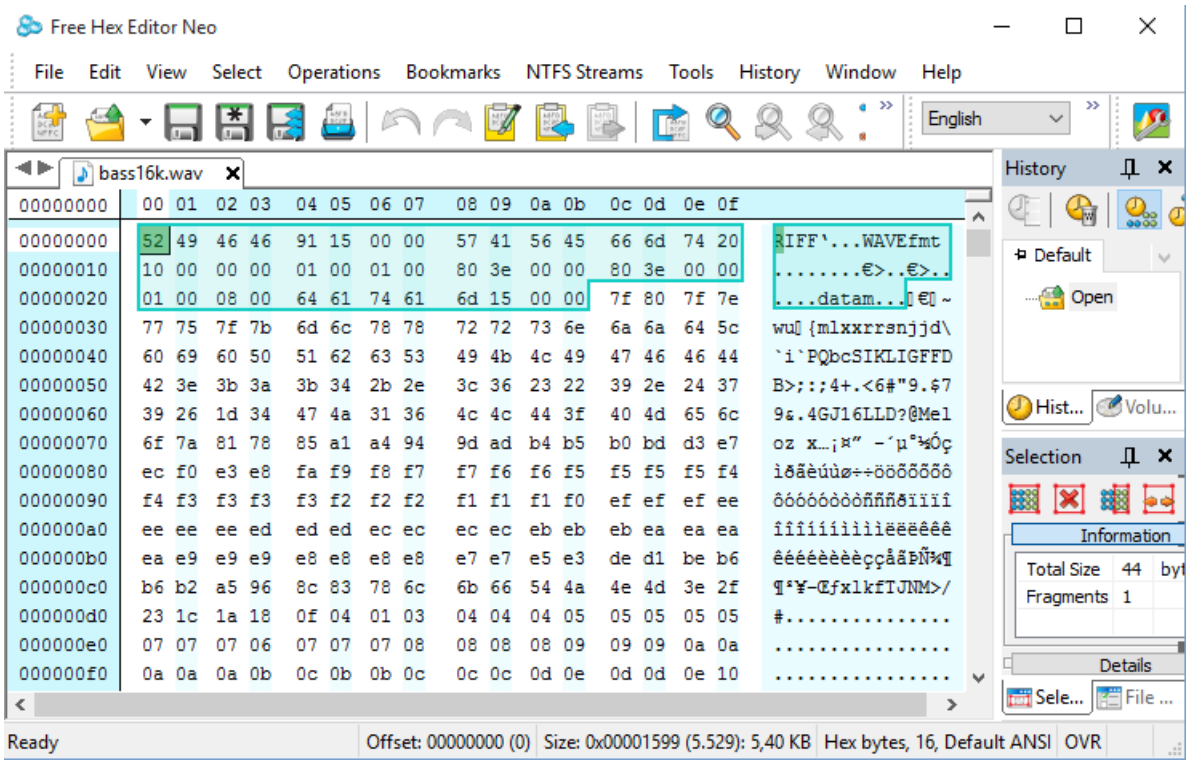
- Arquivo "bass16k.wav": duração de 85ms e tamanho de 1415 *bytes*;
- Arquivo "hihat16k.wav": duração de 208ms e tamanho de 3385 *bytes*;
- Arquivo "snare16k.wav": duração de 194ms e tamanho de 3158 *bytes*.

Depois de realizar o tratamento dos arquivos de áudio, para conferir e confirmar o que foi descrito na Seção 2.6, o arquivo WAVE respectivo ao bumbo foi aberto no *software* gratuito Free Hex Editor Neo. Esse *software* é um editor gratuito que permite visualizar, modificar e analisar arquivos em hexadecimal e binário[6]. A Figura 22 mostra o arquivo de áudio do bumbo aberto no *software* Free Hex Editor Neo e a sequência dos 44 *bytes* do cabeçalho selecionada.

A Figura 23 mostra o cabeçalho do arquivo WAVE em detalhes. Podemos perceber que o cabeçalho está corretamente apresentado conforme as especificações desejadas para o arquivo de áudio: 8 *bits* de codificação PCM, 1 canal de áudio (*mono*) e taxa de amostragem de 16kHz.

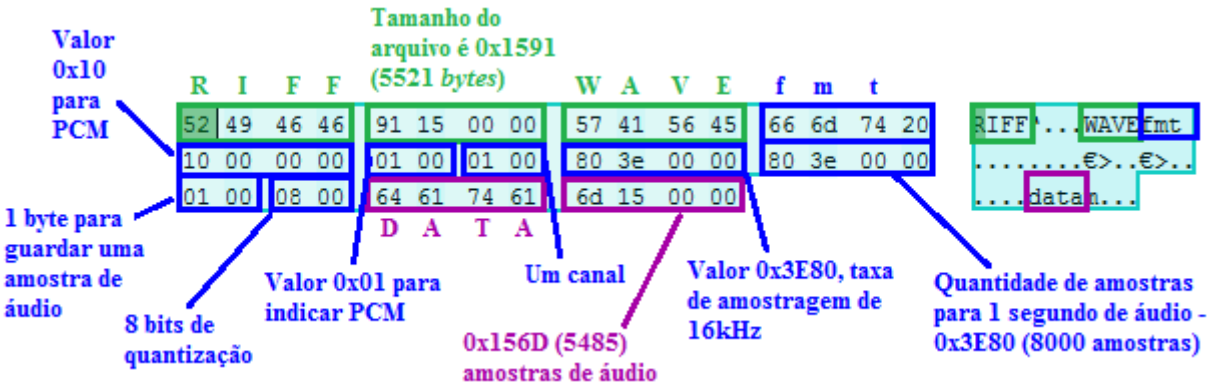
[5] Disponível para *download* em <http://audacityteam.org/download/>. Acesso em nov. 2015.
[6] Disponível para *download* em <http://www.hhdsoftware.com/free-hex-editor>. Acesso em nov. 2015.

Figura 22: Arquivo de áudio aberto no *software* Free Hex Editor Neo.



Fonte: Elaborado pelo autor.

Figura 23: Sequência de *bytes* do cabeçalho do arquivo WAVE.



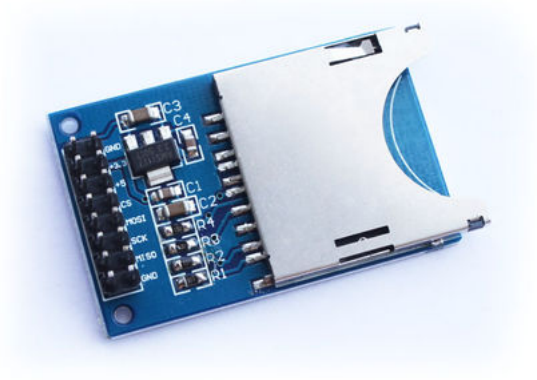
Fonte: Elaborado pelo autor.

3.3.2 Cartão SD

O cartão SD[7] é um cartão de memória que armazena dados em uma memória *flash*. O padrão SD foi desenvolvido pela SD Association[8] no ano 2000 e o nome SD foi dado pois o dispositivo oferece capacidade de criptografia e gestão de direitos autorais. O cartão SD trabalha com tensões de 2,7V à 3,6V e possui dois modos de comunicação, o modo SD e o modo SPI. (SANDISK, 2003)

O ATmega2560 foi programado para utilizar o modo SPI para se comunicar com o cartão SD. Para realizar a interface entre o ATmega2560 e o cartão SD foi utilizado um módulo para cartões SD adquirido no site de compras internacional *eBay*[9]. O módulo pode ser conferido na Figura 24.

Figura 24: Módulo para cartões SD utilizado no sistema.



Fonte: Elaborado pelo autor.

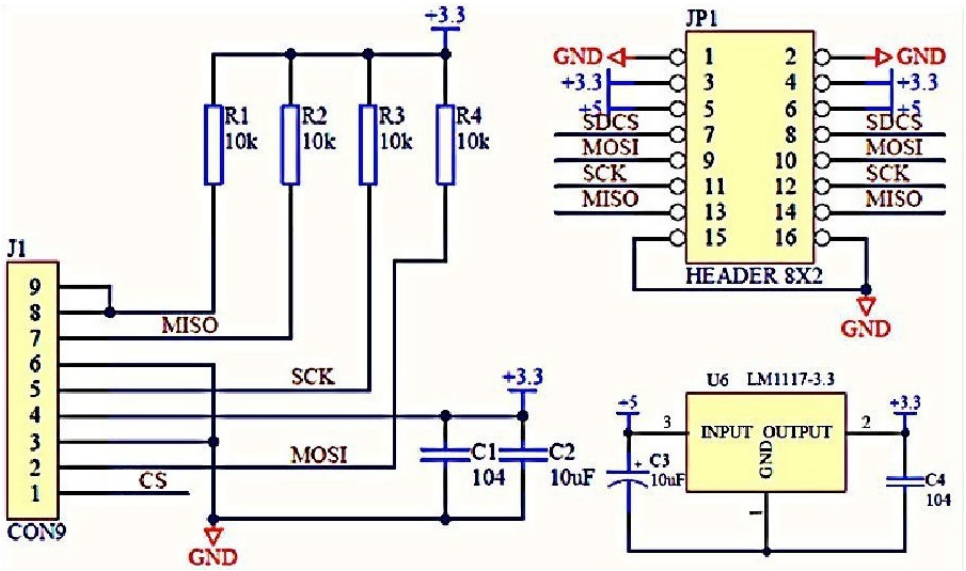
O circuito do módulo do cartão SD utilizado está mostrado na Figura 25, onde os pinos MISO, MOSI, SCK e CS do módulo SD são ligados nos pinos 50, 51, 52 e 53 do ArduinoMega, respectivamente. Percebe-se a partir da Figura 25 que o circuito presente no módulo foi projetado para regular a tensão elétrica de 5 volts do Arduino para 3.3 volts, que é a tensão de trabalho do cartão SD.

3.3.2.1 Leitura dos Arquivos

Durante o decorrer do projeto, tentou-se utilizar algumas bibliotecas genéricas para microcontroladores disponíveis na internet para acesso ao sistema de arquivos FAT16 do cartão SD, não obtendo sucesso na implementação. Partiu-se então para a confecção

[7] Abreviatura do inglês *Secure Digital*
[8] Associação de fabricantes formada em 2000 pela Matsushita (Panasonic), SanDisk e Toshiba
[9] Pode ser acessado em <http://www.ebay.com/>
[10] Disponível em: <http://duino4projects.com/wp-content/uploads/2013/03/Schematic-Arduino-SD-Card-Logging-Shield.jpg>. Acesso em nov. 2015.

Figura 25: Módulo para cartões SD utilizado no sistema.



Fonte: Adaptado do site *duino4projects*[10].

de uma biblioteca própria para acesso aos arquivos e inicialização do cartão SD, porém esse processo se tornou inviável a ser realizado dentro do prazo previsto para a entrega deste projeto de diplomação. Após discussão deste empecilho com o professor orientador, chegou-se a conclusão de passar a utilizar as bibliotecas do Arduino para acesso ao cartão SD. Por consequência, devido à dependência que a biblioteca SD possui com a biblioteca para comunicação SPI do Arduino, esta última também foi utilizada.

Até então, estava sendo utilizada a IDE Atmel Studio 6.2 para o desenvolvimento do projeto, sendo preciso a migração para a IDE Arduino. Pequenas modificações precisaram ser realizadas, não comprometendo o projeto desenvolvido até então. As modificações realizadas encontram-se abaixo.

- Os arquivos *source* escritos em linguagem C que possuíam extensão **.C** precisaram ser convertidos para arquivos de linguagem C++. Isso foi realizado simplesmente alterando a extensão do arquivo de **.C** para **.CPP**. Isso foi necessário pois a IDE do Arduino apresentou erros na compilação dos arquivos **.C**;

- Foi necessária a troca do módulo TIMER3 para o módulo TIMER3 do ATmega2560 pois as bibliotecas do Arduino utilizam o TIMER3 por padrão e este não pode ser utilizado. Devido à similaridade dos módulos temporizadores do ATmega2560, as únicas alterações necessárias foram a troca do nome dos registradores e a troca do nome do vetor de interrupção;

- A última alteração necessária foi a troca do *source* "main.c", que continha a função

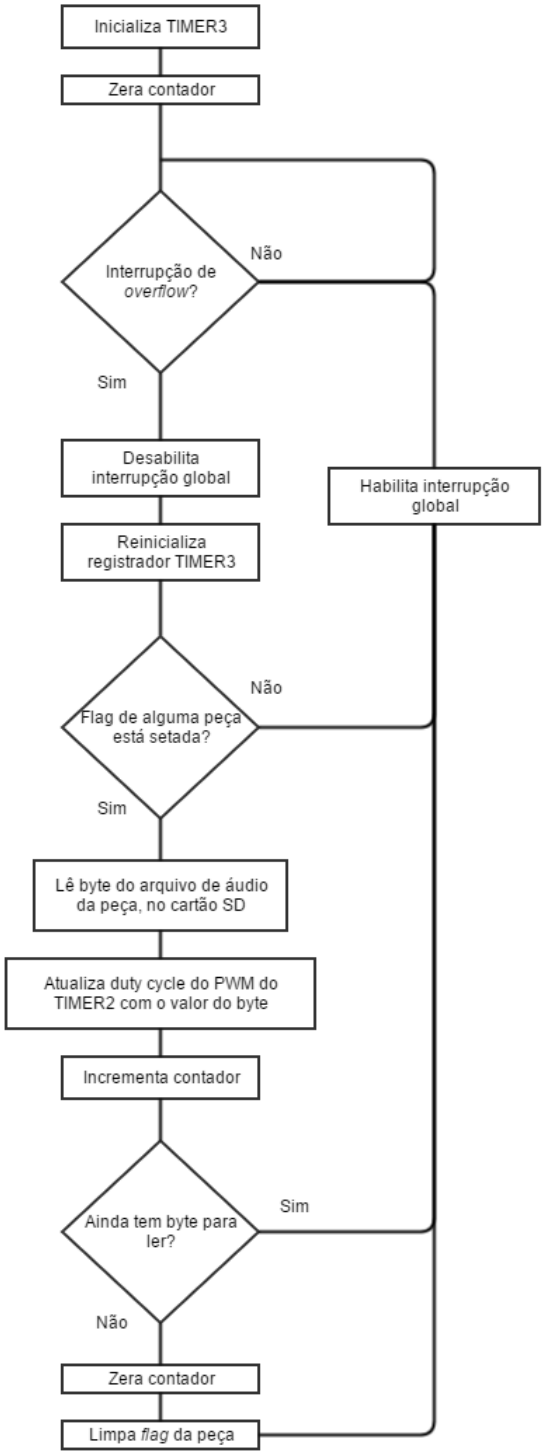
`main()` para um *sketch* do Arduino cuja extensão é **.INO**. O código presente na função `main()` do arquivo "main.c" foi colocado dentro das funções `setup()` e `loop()` no arquivo "WiDrum.ino". Comparando o código de "main.c" com o do arquivo "WiDrum.ino", nota-se que a função `setup()` do Arduino serve para agregar todo o código que está antes do laço infinito `while(1)` e a função `loop()` funciona como o laço infinito, assim agrega o código que está dentro do `while(1)`.

Foram armazenados três arquivos WAVE no cartão SD conforme a configuração mencionada anteriormente. Os arquivos simulam o som do bumbo, do chimbau e a caixa e receberam nomes de "bass16k.wav", "hihat16k.wav" e "snare16k.wav", respectivamente.

A leitura dos arquivos ocorre no momento em que o ATmega2560 recebe do XBee um valor correspondente à alguma peça. Se o primeiro *byte* recebido corresponde à identificação de alguma peça da bateria (0xD1, 0xD2 ou 0xD3), o programa aguarda o recebimento do próximo dado que é o valor da intensidade do impacto. Após o recebimento da informação da intensidade, o cartão SD é acessado, o arquivo referente à peça tocada é aberto e é inicializada a leitura dos *bytes* do arquivo a cada interrupção do TIMER3, ou seja, a cada 62,5ns (16kHz). O valor do *duty cycle* do PWM do módulo TIMER2 é atualizado a cada interrupção do TIMER3 com o valor do *byte* lido. Após o último *byte* do arquivo ser lido, o programa fecha o arquivo e aguarda o recebimento de novos dados.

O fluxograma de execução das tarefas do módulo TIMER3, responsável por controlar a leitura do cartão SD, está apresentado na Figura 26.

Figura 26: Fluxograma de execução das tarefas do módulo TIMER3.



Fonte: Elaborado pelo autor.

3.3.3 Geração do Sinal PWM

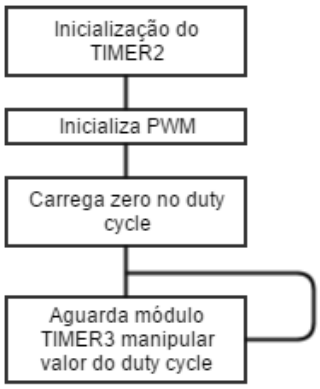
Para a geração do PWM está sendo utilizado o módulo TIMER2 do ATmega2560. Este módulo é muito similar aos outros módulos temporizadores, por isso não será entrado em detalhes de funcionamento do TIMER2.

O TIMER2 tem uma saída PWM que é gerada a partir de um valor determinado no registrador OCR2A. A frequência do PWM foi configurada para ser 62,5kHz, quase 4 vezes a frequência de amostragem do áudio gravado no cartão SD. Experimentalmente, a frequência do PWM ótima seria 10 vezes a frequência de amostragem do áudio, porém, para ter resolução de 8 bits no *duty cycle*, a máxima frequência que pode ser gerada no PWM é 62,5kHz. Essa limitação se deve à frequência de oscilação do sistema, que é de 16MHz ($16MHz/(256bits) = 62,5kHz$).

A cada interrupção do TIMER3, que ocorre na mesma frequência de amostragem do áudio (16kHz), o registrado do *duty cycle* OCR2A do PWM é atualizado com o valor do *byte* lido do arquivo de áudio armazenado no cartão SD. Isso ocorre até o arquivo chegar no fim.

O fluxograma de execução das tarefas do módulo TIMER2, responsável pela geração do PWM, está apresentado na Figura 27.

Figura 27: Fluxograma de execução das tarefas do módulo TIMER2.



Fonte: Elaborado pelo autor.

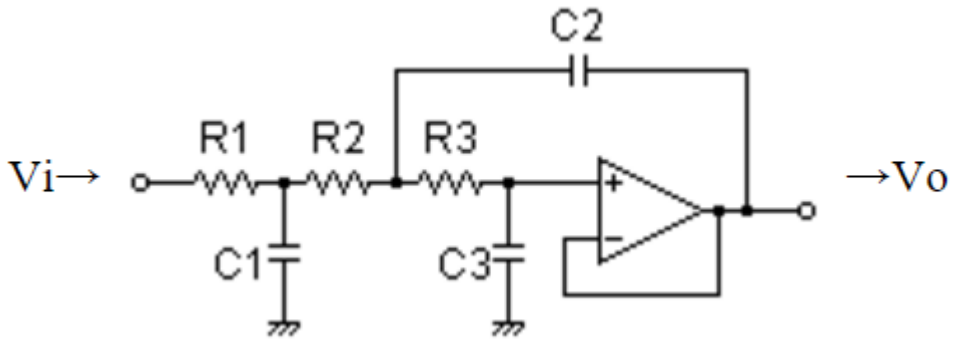
3.3.4 Filtro para Demodulação do Sinal PWM

Para realizar a demodulação do sinal PWM gerado, foi projetado um filtro passa baixas do tipo Chebyshev com frequência de corte de 7,5kHz. A frequência de corte foi escolhida segundo o critério de Nyquist, onde a maior frequência de um sinal deve ser no máximo a metade da frequência da amostragem do sinal. Como o áudio pe amostrado em 16kHz, a frequência de Nyquist é 8kHz, porém utilizou-se 7.5kHz a fim de melhorar a performance do filtro. O tipo Chebyshev foi escolhido pois ele tem uma resposta de atenuação mais rápida, não sendo tão linear na frequência de corte, diferentemente do filtro Butterworth. A não linearidade da resposta do filtro Chebyshev perto da frequência de corte não é relevante pois ela foi projetada para ser de no máximo 1dB, pequena suficiente a fim de não afetar o sinal de áudio.

Foi definido um filtro de terceira ordem, na topologia Sallen-Key, conforme Figura 28. Para projeto do filtro, utilizou-se uma ferramenta de cálculo disponível no sítio virtual japonês *Okawa Denshi*, que oferece diversas ferramentas para cálculos de filtros. Através da configuração da frequência de corte desejada e *overshoot*, a ferramenta forneceu os seguintes valores para os componentes: $R1 = 8.2k\Omega$, $R2 = 100k\Omega$, $R3 = 33k\Omega$, $C1 = 4,7nF$, $C2 = 1,8nF$ e $C3 = 68pF$. O filtro projetado possui a função de transferência dada na Equação (3.3). A resposta em frequência simulada deste filtro está mostrada na Figura 29. Percebe-se que na frequência de corte a resposta tem um *overshoot* de 1dB, confirmando o *ripple* de ganho projetado. Não foram realizados experimentos para verificação da resposta em frequência experimental do filtro pois o tempo limite para a entrega da monografia foi atingido antes.

$$G(s) = \frac{6,42.10^{13}}{s^3 + 50465,3s^2 + 3,09.10^9 s + 6,42.10^{13}} \tag{3.3}$$

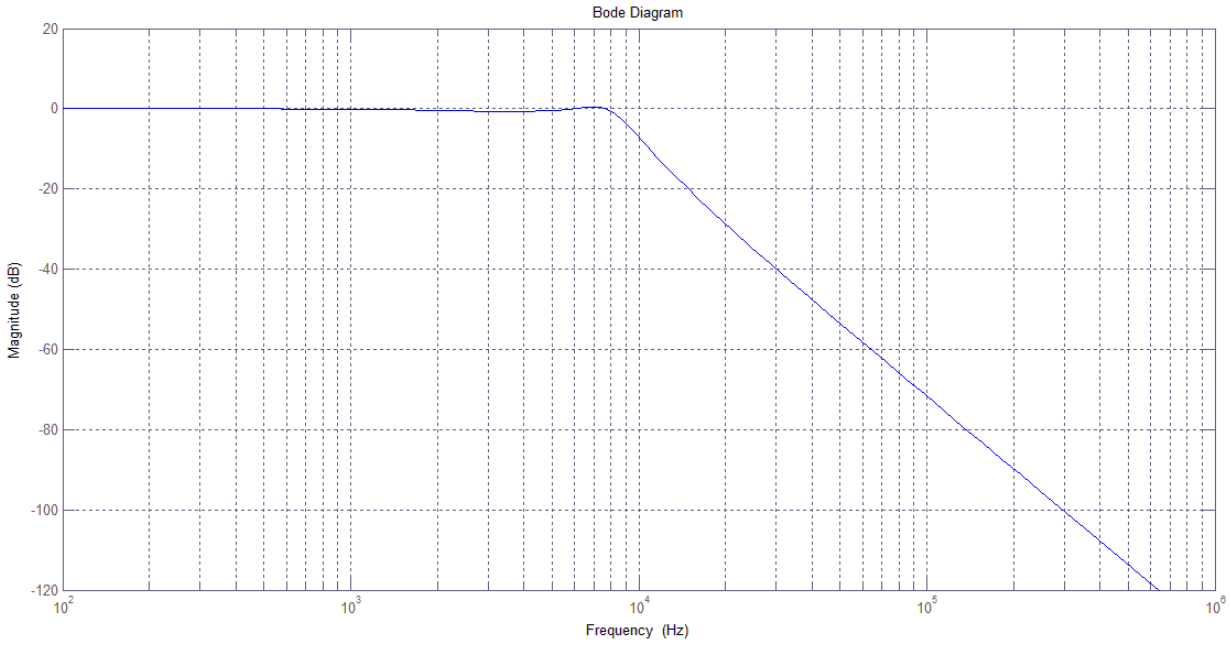
Figura 28: Topologia do filtro de terceira ordem para demodulação do PWM.



Fonte: Adaptado do site *Okawa Denshi*[11].

[11] Disponível em: <http://sim.okawa-denshi.jp/en/Sallenkey3Lowkeisan.htm>. Acesso em nov. 2015.

Figura 29: Resposta em frequência simulada do filtro de projetado.

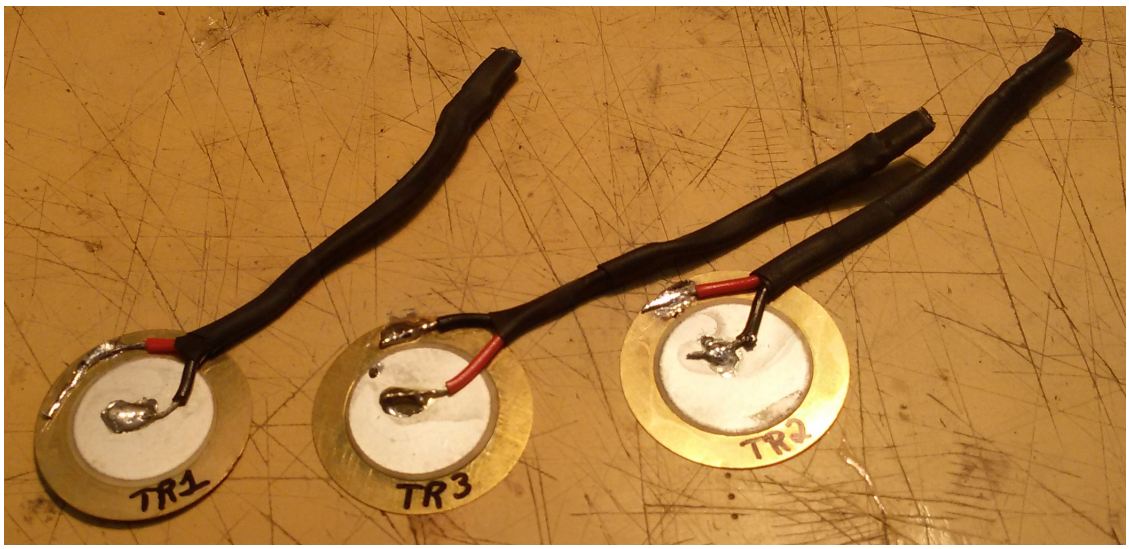


Fonte: Elaborado pelo autor.

3.4 Transdutor Piezoelétrico

O transdutor utilizado (Figura 30) é vendido em lojas de eletrônica do centro da cidade de Porto Alegre (RS) não possuindo *datasheet* ou qualquer especificação, inclusive sobre o fabricante. Tentativas de descobrir na Internet algum fabricante que disponibilizava sensores similares não obtiveram sucesso. Para realizar o projeto do condicionador segundo o modelo adotado na Seção 2.5.1, o qual desconsidera as constantes piezoelétricas do material, foi preciso descobrir o valor da capacitância do sensor. Foi utilizado um medidor RLC da fabricante Leader e modelo LCR-745 para a medição das capacitâncias dos três transdutores piezoelétricos utilizados já com os fios soldados.

Figura 30: Identificação dos transdutores.



Fonte: Elaborado pelo autor.

Os transdutores foram identificados aleatoriamente como TR1, TR2 e TR3, conforme Figura 30. Tomou-se cuidado para trançar os fios a fim de reduzir as interferências magnéticas (EMI). Na Tabela 1 estão informações das características de medição do LCR-745 para capacitâncias utilizando o modo de leitura paralelo e frequência de operação de 1kHz, segundo (HIGASHI, 1990). Adotou-se o modo de medição paralelo e com frequência de 1kHz do LCR-745 pois é o que apresenta menor incerteza nas medidas dentro da faixa de valores desejada (1nF até 100nF), segundo (HIGASHI, 1990).

Na Tabela 2 estão apresentados os valores encontrados para as capacitâncias dos transdutores. Realizando a média dos valores encontra-se 26,9nF. Adotou-se um valor de 27nF para realizar o projeto do condicionador pois é um valor de capacitância comercial e difere pouco do valor encontrado.

A tensão elétrica de saída dos sensores piezoelétricos podem variar de microvolts até centenas de volts, requerendo diferentes possibilidades de condicionamento. Ao designar

Tabela 1: Características de medição do LCR-745 para cargas capacitivas no modo paralelo e frequência de 1kHz.

Escala	Resolução	Exatidão
200pF	0,1pF	$\pm(1\% \pm 3d)$
2000pF	1pF	$\pm(0,35\% \pm 2d)$
20nF	0,01nF	$\pm(0,35\% \pm 2d)$
200nF	0,1nF	$\pm(0,35\% \pm 2d)$
2uF	0,001uF	$\pm(0,35\% \pm 2d)$

Tabela 2: Capacitâncias dos transdutores piezoelétricos utilizados.

Transdutor	Capacitância
TR1	$26,9nF \pm 0,35\%$
TR2	$26,5nF \pm 0,35\%$
TR3	$27,3nF \pm 0,35\%$

o condicionador, deve-se considerar a frequência de operação, a amplitude do sinal e a impedância de entrada. (KARKI, 2000)

3.4.1 Condicionamento

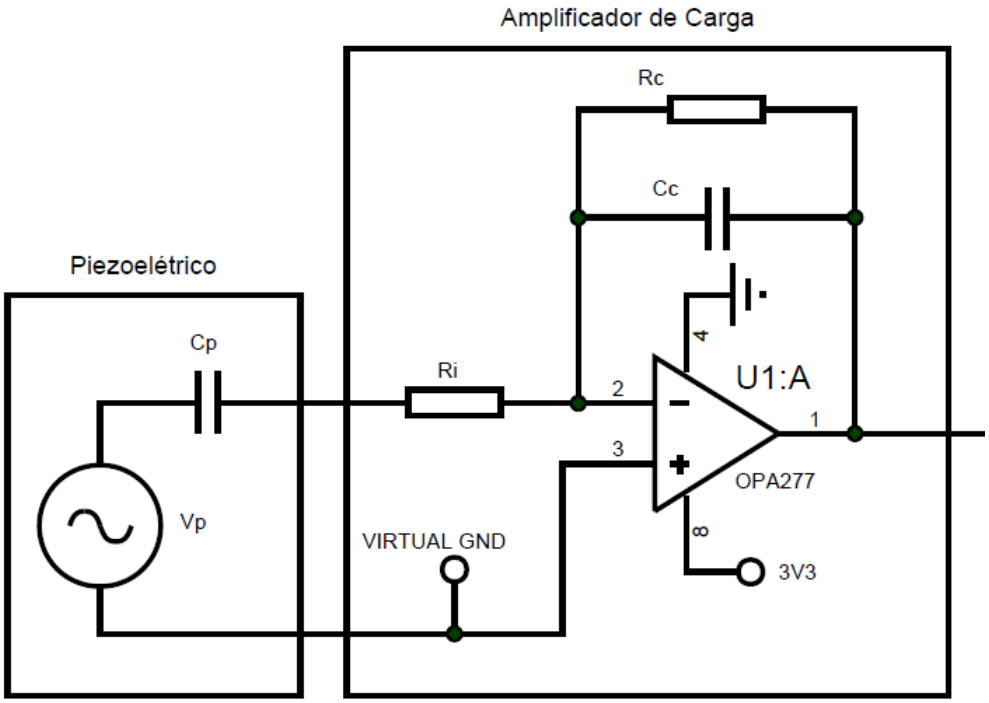
Como já foi discutido, a saída do sensor piezoelétrico é em carga elétrica. Dessa forma, é preciso aplicar um condicionamento adequado, pois a medida é realizada em tensão elétrica. Utilizou-se para isso um amplificador de carga. A Figura 31 mostra o circuito projetado para o condicionador.

Como a saída do condicionador irá oscilar entre valores positivos e negativos, foi adicionado um circuito de terra virtual como mostrado na Figura 32.

A função de transferência resultante para o circuito condicionador é dada pela Equação (3.4). A capacitância do sensor definida como 27nF a partir das medições obtidas com o medidor RLC está sendo considerada na modelagem.

$$\frac{V_c}{V_p} = \frac{-s.\dfrac{1}{R_i.C_c}}{\left(\dfrac{1}{R_i.C_p} + s\right)\left(\dfrac{1}{R_c.C_c} + s\right)} \tag{3.4}$$

Figura 31: Condicionamento do sensor piezoelétrico.



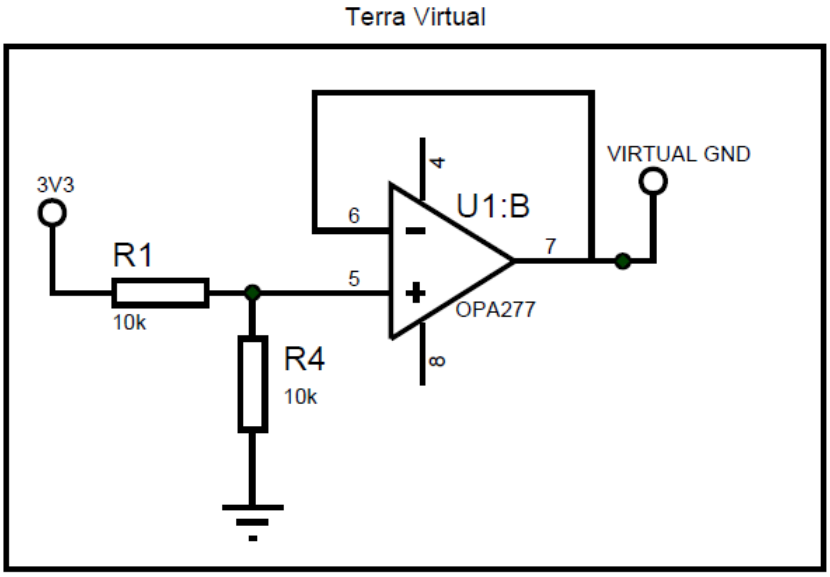
Fonte: Elaborado pelo autor.

Onde Vp é a tensão do piezoelétrico, Vc é a tensão na saída do amplificador de carga, Cp é a capacitância do piezoelétrico, Ri é a resistência de entrada, Rc e Cc são a resistência e a capacitância do amplificador de carga respectivamente. Devida à alta impedância dos sensores piezoelétricos, deve ser utilizado um amplificador com alta impedância de entrada e baixa tensão de *offset*. O amplificador operacional utilizado para na configuração do condicionador é o OPA277 da fabricante Texas Instruments, o qual apresenta impedância de entrada em modo comum de 250GΩ e tensão de *offset* máxima de $\pm 20\mu V$. Como a impedância de entrada do OPA277 é alta e sua tensão de *offset* é pequena, podemos considerar que o componente não influenciará no equacionamento.

O amplificador de carga da Figura 31 irá equilibrar a carga injetada na entrada negativa do amplificador através do carregamento do capacitor de realimentação Cc. O resistor Rc descarrega o capacitor Cf em uma taxa lenta a fim de prevenir a saturação do amplificador. O resistor Rf também serve para realizar a polarização DC da entrada negativa do amplificador. Na configuração de amplificador de carga proposta, o circuito se comporta como um filtro passa-faixa.

Segundo (KARKI, 2000), a frequência de corte inferior do amplificador de carga é determinada por Rc e Cc e a frequência de corte superior é determinada por Ri e Cp,

Figura 32: Circuito do terra virtual.



Fonte: Elaborado pelo autor.

conforme Equação (3.5) e Equação (3.6) respectivamente.

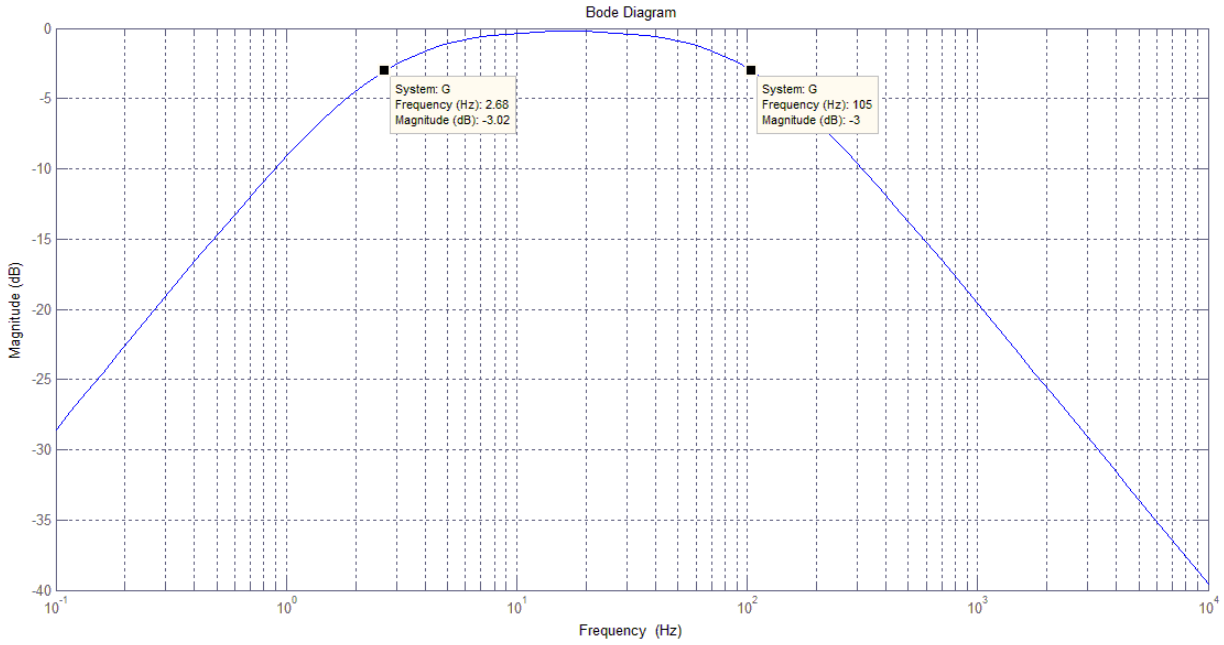
$$f_L = \frac{1}{2\pi R_c.C_c} \tag{3.5}$$

$$f_H = \frac{1}{2\pi R_i.C_p} \tag{3.6}$$

Escolheram-se frequências de corte de forma que a faixa de passagem contemplasse as frequências dos impactos de um baterista. Considerou-se como frequência mínima 0,2Hz e frequência máxima de 100Hz. Foram definidos os valores comerciais $Ri = 56k\Omega$ e $Rc = 2,2M\Omega$ e com estes valores obteve-se $f_L = 2,67Hz$ e $f_H = 105,26Hz$. A resposta em frequência do sensor piezoelétrico não é conhecida, com isso não se pode afirmar que o sensor está atuando corretamente na frequência projetada de 2,67Hz, visto que cristais piezoelétricos não têm resposta linear para baixas frequências (BALBINOT; BRUSAMARELLO, 2011). Como a carga do piezoelétrico é transferida para o capacitor Cc, o valor de Cc determina o ganho do amplificador de carga, assim foi escolhido $Cc = 27nF$ para ganho unitário. Com estes valores a função de transferência fica como mostrado na Equação (3.7) e sua resposta em frequência simulada está apresentada na Figura 33.

$$\frac{V_c}{V_p} = \frac{-661,37s}{s^2 + 678,21s + 11134,27} \tag{3.7}$$

Figura 33: Resposta em frequência simulada do condicionador para o sensor piezoelétrico.



Fonte: Elaborado pelo autor.

Na saída do condicionador foi adicionado um estágio de filtro passa baixa passivo para evitar o efeito de *aliasing*, como mostrado na Figura 34. Como a frequência de amostragem do conversor A/D do PIC18F2550 é de 5kHz e segundo o teorema de *Nyquist* a máxima frequência do sinal de entrada deverá ser 2,5kHz, foi projetado um filtro passa baixas passivo de segunda ordem com atenuação de 40dB na frequência de corte de 2,25kHz. Foi escolhido um filtro passivo pois ele atua adequadamente na frequência de corte projetada, visto que as frequências de interesse são aproximadamente 25 vezes menor que a frequência de corte do filtro. Considerando a tensão entrada de R1 como Vi e a tensão de saída de R2 como Vo, a Equação (3.8) mostra a função de transferência do filtro anti *aliasing*.

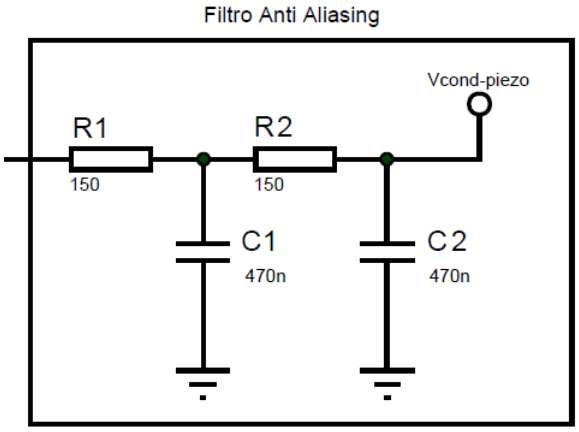
$$\frac{V_i}{V_o} = \frac{1}{1 + s(R1.C1 + R1.C2 + R2.C2) + s^2(R1.C1.R2.C2)} \tag{3.8}$$

Para a frequência de corte de 2,25kHz, os valores dos componentes escolhidos foram $R1 = R2 = 150\Omega$ e $C1 = C2 = 470nF$. A função de transferência já com os valores utilizados mostrada na Equação (3.9) tem a resposta em frequência simulada da Figura 35.

$$\frac{V_i}{V_o} = \frac{1}{1 + 2,11.10^{-4}s + 4,97.10^{-9}s^2} \tag{3.9}$$

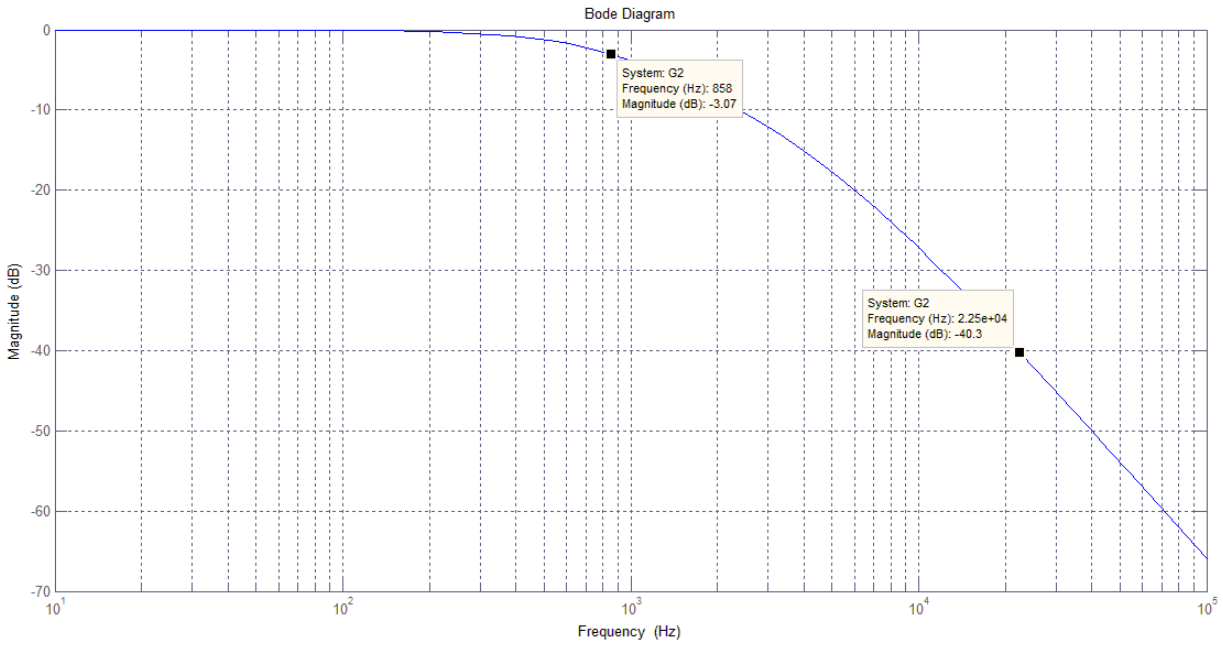
O circuito completo do condicionador para o piezoelétrico encontra-se na Figura 36.

Figura 34: Circuito do filtro anti *aliasing*.



Fonte: Elaborado pelo autor.

Figura 35: Resposta em frequência simulada do filtro anti *aliasing*.



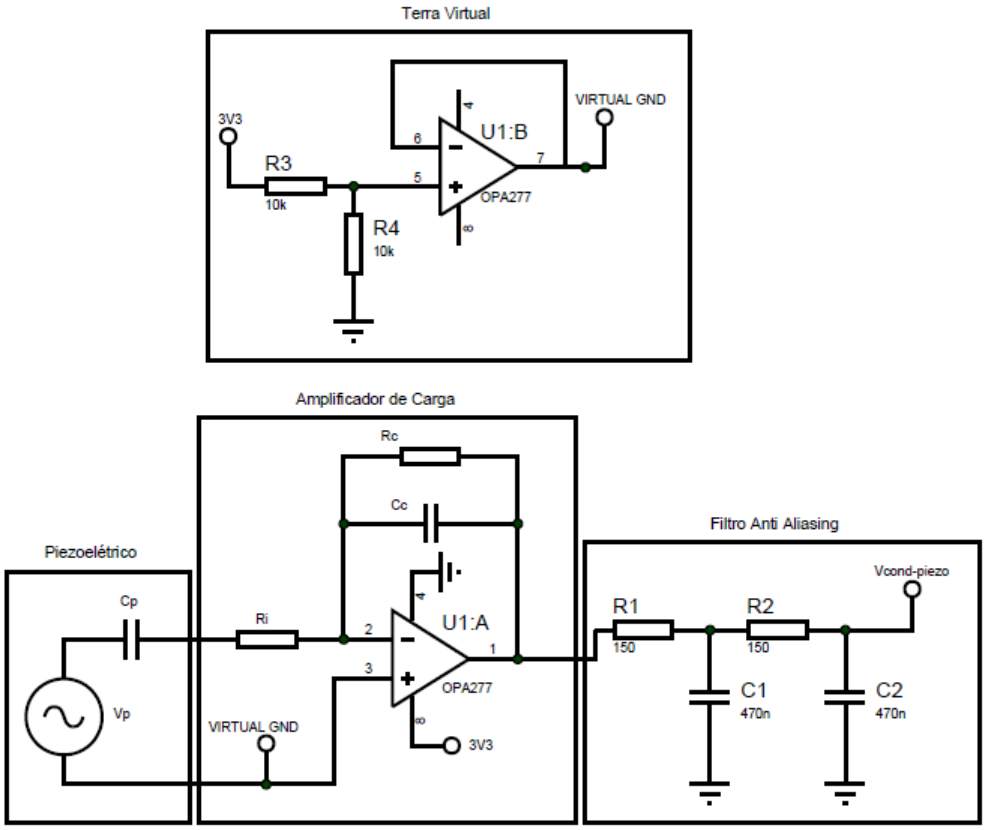
Fonte: Elaborado pelo autor.

Sua função de transferência está descrita na Equação (3.10) e a resposta em frequência na Figura 37.

$$\frac{V_c}{V_p} = \frac{-661,37s}{4,97.10^{-9}s^4 + 2,11.10^{-4}s^3 + 1,143s^2 + 680,6s + 1113} \tag{3.10}$$

Percebe-se que o filtro anti *aliasing* não interfere significativamente na resposta em

Figura 36: Circuito do condicionador completo.



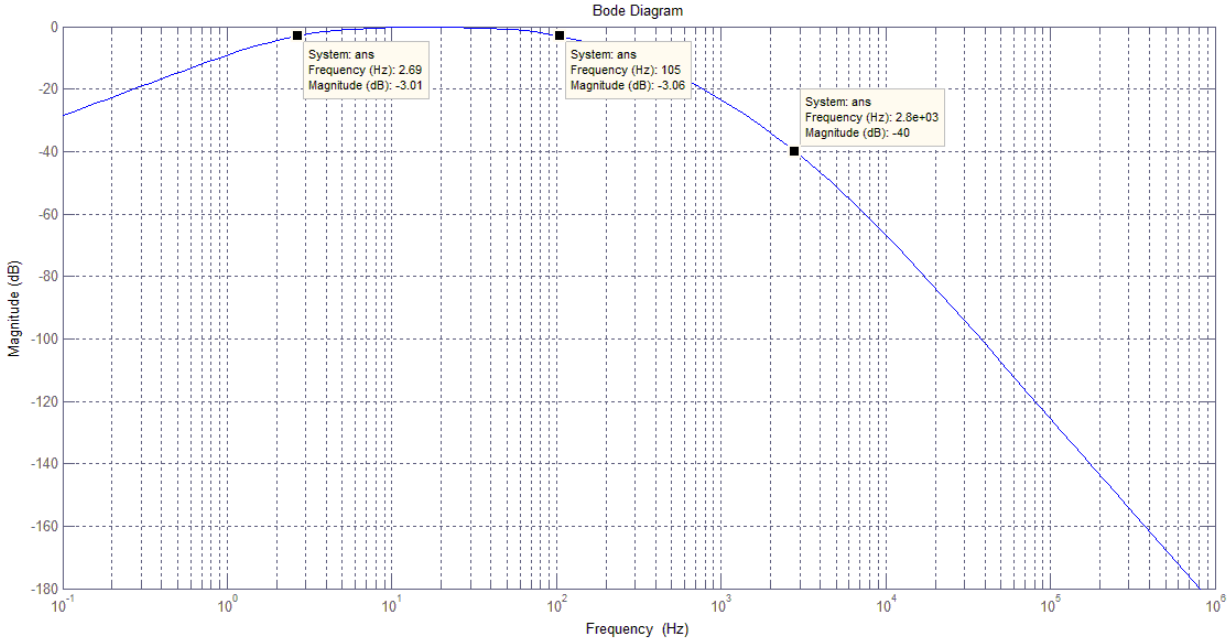
Fonte: Elaborado pelo autor.

frequência simulada do condicionador de carga e que na faixa de frequências desejada o ganho é praticamente constante. Após passar pelo circuito de condicionamento, o sinal é adquirido pelo módulo conversor A/D do microcontrolador PIC18F2550.

Como o sinal do condicionador está sendo adquirido no microcontrolador PIC18F2550 com 8 *bits* de quantização e a tensão de referência do seu conversor A/D é de 3,3V, tem-se a resolução mostrada na Equação (3.11).

$$Resolução = \frac{3,3-0}{2^8} = \frac{3,3}{256} = 12,89mV \tag{3.11}$$

Figura 37: Resposta em frequência simulada do condicionador completo.



Fonte: Elaborado pelo autor.

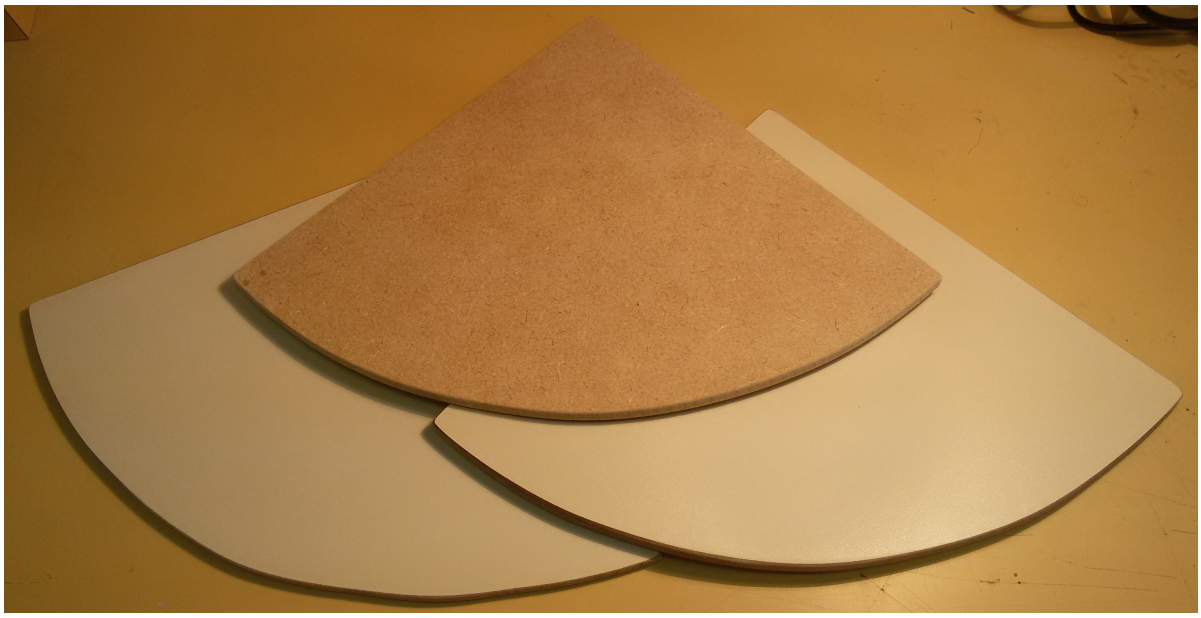
3.4.2 Plataformas das Peças

Foram construídas três plataformas para a simulação das peças da bateria, uma para cada transdutor.

A plataforma foi construída em quatro estágios e sua constituição é de material MDF com espessura de 8 milímetros e forma geométrica de um quarto de círculo com diâmetro de 25 centímetros. A forma geométrica foi escolhida de forma a proporcionar facilidade na hora de realizar um impacto pelo usuário, de forma similar aos pratos de bateria acústicos.

O primeiro estágio da fabricação dos *pads* consiste no corte do MDF na forma de quarto de círculo. Na figura Figura 38 é possível identificar as plataformas no primeiro estágio de fabricação.

Figura 38: Plataformas para ensaios de impacto.



Fonte: Elaborado pelo autor.

No segundo estágio foi feito o revestimento da face onde são realizados os impactos com EVA (espuma vinílica acetinada) de espessura de 5 milímetros, material bastante flexível, para absorver o impacto e proporcionar uma resposta adequada para o usuário. O EVA foi escolhido pelo autor do projeto pois foi o material com maior custo benefício e que mais se assemelha à resposta de uma bateria acústica.

A colagem do EVA foi feita com adesivo PVA (poliacetato de vinila) e tanto o EVA quanto o adesivo foram encontrados em lojas de artesanato. Na Figura 39 pode se verificar em detalhe um *pad* revestido.

Figura 39: Plataformas revestidas com EVA.

Fonte: Elaborado pelo autor.

O terceiro estágio foi a realização de furos para fixação na estrutura mostrada na Figura 40. A estrutura consiste em um suporte para pratos de bateria que foi fixado em uma morsa.

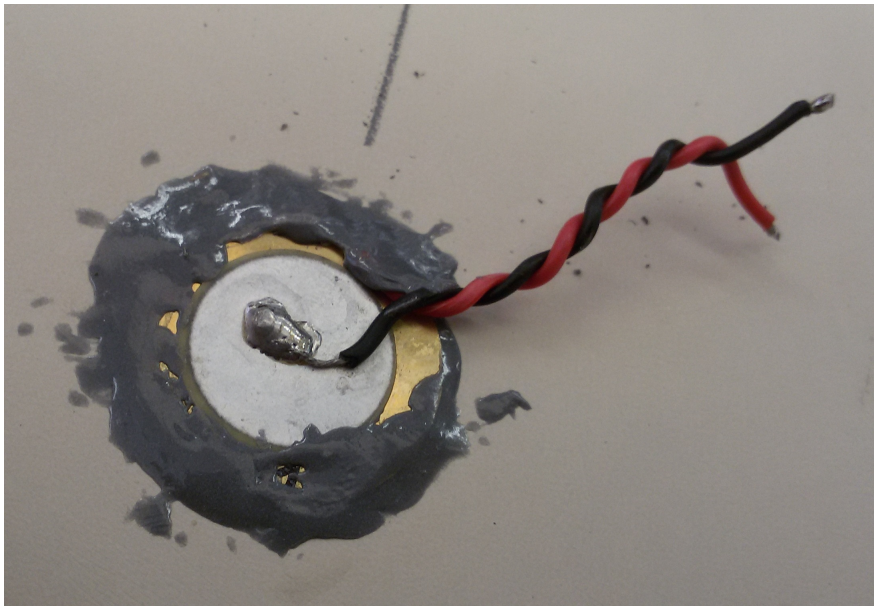
Figura 40: Estrutura de fixação da plataforma.

Fonte: Elaborado pelo autor.

O quarto e último estágio de confecção das plataformas foi a fixação dos transdutores em suas respectivas plataformas, conforme Figura 41. Os transdutores foram cimentados

com solda plástica (conhecida como Poxipol) à uma distância de 17 centímetros do centro do quarto de círculo, sendo esta distância escolhida pois está mais próxima da região que comumente são realizados os impactos.

Figura 41: Piezoelétrico cimentado à plataforma.



Fonte: Elaborado pelo autor.

3.4.3 Ensaio de Impacto

Este ensaio não se destinou à calibração do sensor piezoelétrico, apenas para verificar o funcionamento do piezoelétrico na faixa de frequências desejada, visto que não se obteve informações sobre este sensor.

Para obter a forma do sinal e realizar uma validação do transdutor foram realizados ensaios de impacto do sensor nas plataformas de madeira construídas juntamente com acelerômetros de alta qualidade disponibilizados pelo Laboratório de Instrumentação Eletro-Eletrônica da universidade. Os sensores utilizados foram acelerômetros de modelo Triaxial DeltaTron Accelerometer Type 4520 da fabricante Brüel & Kjæer. O procedimento realizado foi efetuar medições simultâneas com dois acelerômetros e o sistema desenvolvido para posteriormente analisar e comparar as formas dos sinais obtidos e as frequências medidas.

Um acelerômetro foi fixado com fita dupla face na parte de cima da plataforma, afastado da região de onde são realizados os impactos, e outro acelerômetro foi fixado também com fita dupla face na parte de baixo da plataforma, bem próximo ao sensor piezoelétrico. A fixação pode ser conferida na Figura 42 e Figura 43.

Figura 42: Acelerômetro fixado na parte de cima da plataforma.

Fonte: Elaborado pelo autor.

O acelerômetros utilizados medem aceleração em três eixos, porém só se utilizou a medida do eixo Z, o qual é paralelo ao eixo normal do piezoelétrico. A sensibilidade informada para o eixo Z pelo fabricante para o acelerômetro colado na parte de cima da estrutura é de $9,817mV/g$ e para o acelerômetro colado embaixo da estrutura é de $10,82mV/g$. As sensibilidades foram utilizadas para se obter os valores em aceleração. Os sinais dos acelerômetros foram adquiridos através da placa de aquisição de dados NI 9234 acoplada na estrutura cDAQ-9174, ambas da fabricante National Instruments. Para a comunicação das placas de aquisição com o computador, utilizou-se o *software* LabVIEW 2015 Student Version, o qual foi criado pela mesma fabricante das placas. A placa de aquisição de dados utilizada tem capacidade máxima de aquisição de 51,2 kS/s em quantização de 24 *bits*, porém foi utilizada uma taxa de amostragem de 50kHz evitando trabalhar na capacidade máxima.

O microcontrolador PIC18F2550 foi utilizado para adquirir o sinal analógico proveniente do condicionador do piezoelétrico ao mesmo tempo que foram adquiridos os sinais dos acelerômetros. Foi utilizada uma taxa de 5kHz para aquisição no microcontrolador e quantização de 8 *bits*.

Os sinais dos acelerômetros foram adquiridos em um computador diferente de onde foram adquiridos os sinais do piezoelétrico. Ambos sistemas de aquisição realizaram medidas durante 5 segundos, tempo suficiente para realizar um impacto na plataforma. As aquisições foram iniciadas manualmente, sendo posteriormente ajustados no *software* LabVIEW 2015 para o impacto ter início ao mesmo tempo nas duas aquisições, visto que

Figura 43: Acelerômetro fixado na parte de baixo da plataforma.

Fonte: Elaborado pelo autor.

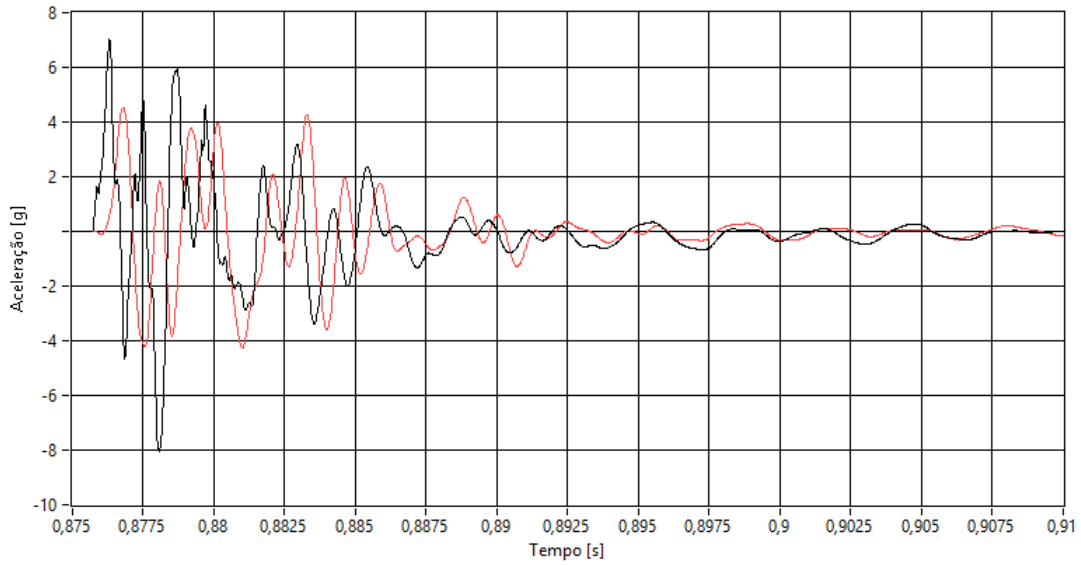
a região de interesse é pequena comparada aos 5 segundos de aquisição.

Em uma primeira análise das formas de onda obtidas dos acelerômetros, disponíveis na Figura 44 e Figura 45, observou-se que a forma de onda do sinal do acelerômetro colado embaixo da estrutura apresentou semelhança à forma de onda do sinal do acelerômetro colado em cima da estrutura, porém com amplitude um pouco menor e um leve atraso. Na Figura 45 é possível perceber alguns picos de amplitude similar nos sinais dos dois acelerômetros. Para o caso da Figura 44, a amplitude da aceleração transmitida para a parte de baixo comparada com a amplitude da aceleração de excitação, considerada na parte de cima, apresentou uma atenuação do sinal de 2,48%.

Esta análise inicial indica não há atenuação significativa na vibração transmitida da parte de cima da plataforma para a parte de baixo. O atraso visualizado no sinal do acelerômetro fixado embaixo é aproximadamente 400us e não é relevante para o sistema. Uma análise mais profunda da estrutura montada, em termos de transmissibilidade e frequência de ressonância, não foi realizada pois foge ao escopo deste projeto, ficando a ideia de um trabalho futuro. A partir de então se considerou apenas o sinal de aceleração obtido na parte de baixo.

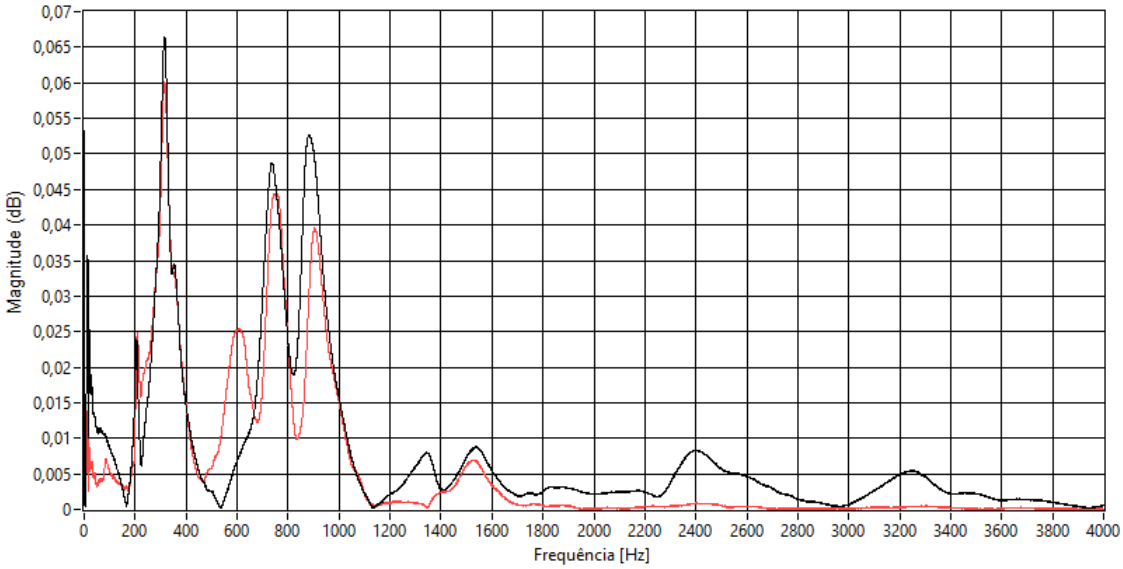
Uma vez que o sinal obtido do condicionador do piezoelétrico tem resposta de um filtro passa faixas, deve-se considerar os sinais do acelerômetro para a mesma faixa de frequência, ou seja, criar um filtro com as mesmas frequências de corte e mesma resposta para o sinal do acelerômetro. Para implementação deste filtro, utilizou-se de rotinas para filtros digitais no *software* LabVIEW, simulando um filtro passa alta na frequência de

Figura 44: Aquisições iniciais dos acelerômetros.



Fonte: Elaborado pelo autor.

Figura 45: FFT das aquisições iniciais dos acelerômetros.



Fonte: Elaborado pelo autor.

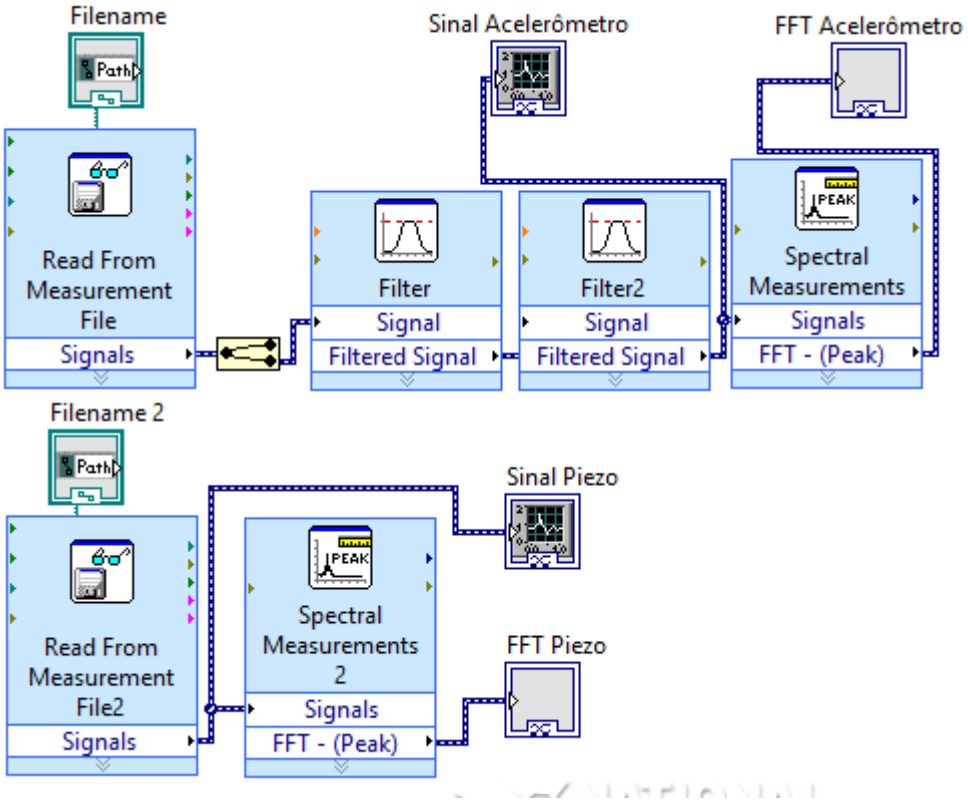
corte de 2,67Hz e um filtro passa baixa na frequência de corte de 105,26Hz, ambos do tipo Butterworth de ordem 1, que possuem a resposta em frequência mais similar à resposta em frequência do amplificador de carga. Os filtros podem ser conferidos na rotina da Figura 46.

Os ensaios realizado foram feitos com apenas um impacto na plataforma e também

com uma série de impactos com frequência de 16Hz, o que no contexto musical seria a execução de um compasso musical divido em fusas com um tempo de 120 impactos por minuto. Isso é considerado uma velocidade rápida para execução, não sendo todos bateristas que conseguem executá-la.

Os valores obtidos pelo microcontrolador foram manipulados no *software* LabVIEW 2015 juntamente com os valores dos acelerômetros. A rotina para plotagem dos gráficos e implementação dos filtros digitais encontra-se na Figura 46.

Figura 46: Rotina de LabVIEW para a plotagem dos sinais do acelerômetro e do piezoelétrico.



3.5 Gravação da execução por MIDI

A gravação da execução do usuário foi realizada com um computador, via protocolo MIDI. A comunicação em MIDI é feita de maneira serial e computadores modernos não possuem mais portas seriais nem portas MIDI, somente portas USB. Para resolver este problema de forma simples e rápida, devido à limitação de tempo para elaboração do projeto de diplomação, resolveu-se utilizar um dispositivo conversor de sinais MIDI para USB ao invés de desenvolver o circuito e a lógica necessária no próprio circuito do módulo de processamento. O conversor MIDI para USB utilizado está mostrado na Figura 47 e foi adquirido no site de compras internacional *eBay*[12].

Figura 47: Conversor MIDI para USB utilizado.

Fonte: Elaborado pelo autor.

O conversor MIDI para USB é do tipo *plug and play*, ou seja, o dispositivo é reconhecido e configurado automaticamente pelo sistema operacional, que no caso foi utilizado o Windows 10 onde já possui um *driver* para o conversor.

Uma vez instalado o conversor MIDI para USB, é necessário um *software* para adquirir os sinais provenientes da bateria. Para isso, utilizou-se a estação de trabalho de áudio digital (DAW[13]) Nuendo, da fabricante Steinberg, que possibilita a gravação, edição e execução de áudios digitais.

[12] Pode ser acessado em <http://www.ebay.com/>
[13] Abreviação do inglês *Digital Audio Workstation*

Para ter acesso aos sinais provenientes do conversor com maior qualidade e menor latência, foi utilizado um *driver* ASIO instalado no computador. A latência mencionada se refere ao atraso entre o envio de uma informação de áudio por um aplicativo e este ser reproduzido pela placa de som, ou sinais de entrada do dispositivo de áudio estarem disponíveis para o aplicativo de áudio.

A sigla ASIO significa *Audio Stream Input/Output* e se refere à um protocolo de especificações para *drivers* de placas de som de computadores. O protocolo ASIO foi desenvolvido pela fabricante Steinberg e possibilita o acesso direto ao *hardware* de som, externo ou interno. O *driver* ASIO utilizado foi o *ASIO4ALL*[14], que é gratuito e foi desenvolvido para ser compatível com qualquer dispositivo de áudio não sendo preciso um *driver* proprietário. Por esta razão, o *ASIO4ALL* é um *driver* ASIO universal.

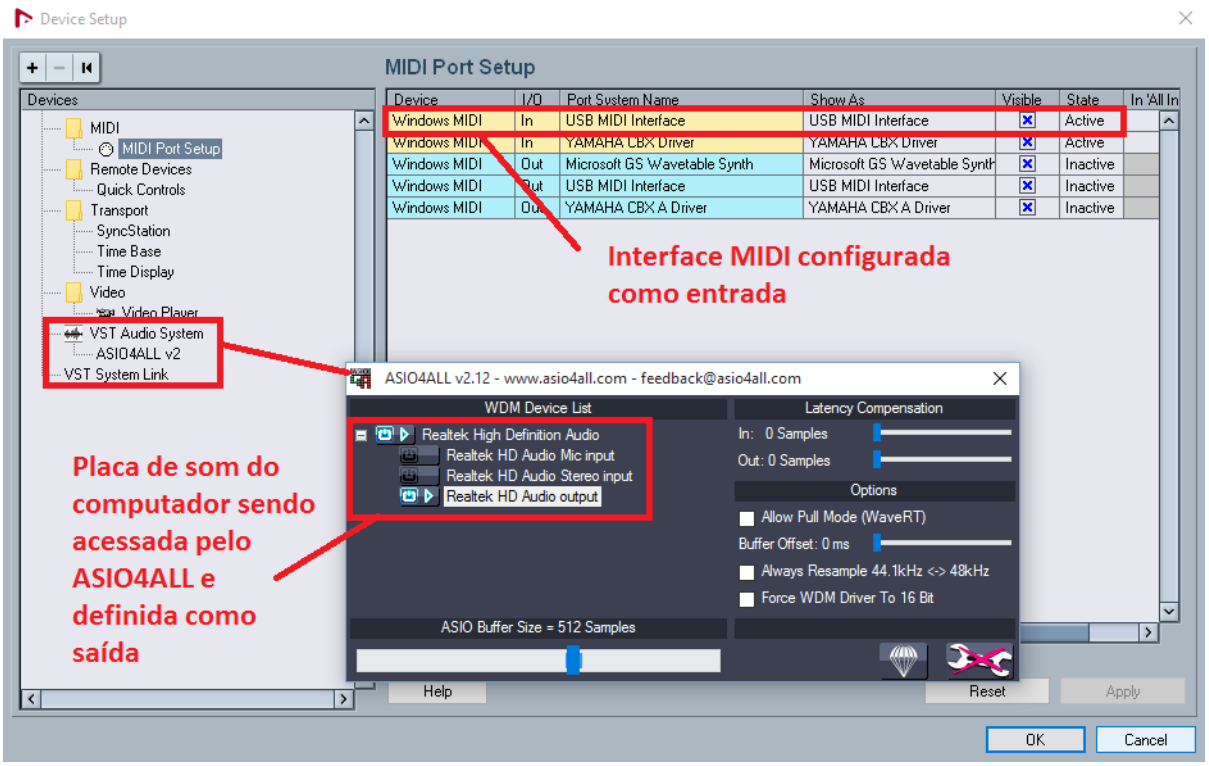
Os *drivers* ASIO ignoram o caminho normal do áudio entre um *hardware* e um programa de gravação através de camadas de *software* intermediárias do sistema operacional do Windows, assim um aplicativo se conecta diretamente à placa ou dispositivo de som. A função de um *driver* ASIO pode ser confirmada quando se utiliza uma DAW configurada com um *driver* ASIO para acesso à placa de som interna do computador. Quando a DAW estiver em funcionamento, o sistema de som padrão do Windows não irá funcionar pois o DAW estará com acesso direto à placa de som. Neste caso, se utilizarmos ao mesmo tempo um aplicativo que utilize o sistema de som padrão do Windows, como por exemplo o *Windows Media Player*, não obteremos sucesso pois a placa de som está disponível exclusivamente para a DAW. Na Figura 48 é possível identificar a interface MIDI para USB configurada como entrada do sistema de áudio e a placa de som do computador sendo acessada diretamente pelo *driver* ASIO4ALL.

Para o envio das mensagens MIDI do bloco de processamento central para o computador, foi utilizado o módulo USART0 do ATmega2560 no qual está descrito na Subseção 3.2.1.4. As mensagens MIDI utilizadas estão descritas na Tabela 3. Como existem três peças na bateria eletrônica, foi utilizado um valor de nota MIDI para cada peça no *Data Byte* 1, sendo 0b00100100 (36 em decimal) a nota respectiva ao bumbo, 0b00101000 (40 em decimal) a nota respectiva à caixa e 0b00101110 (46 em decimal) a nota respectiva ao chimbau. O valor do *Data Byte* 2 é referente ao valor da intensidade do impacto realizado pelo usuário, sendo este valor a intensidade adquirida do sensor piezoelétrico.

Para poder escutar a gravação realizada pelo usuário com sons simulados de uma bateria acústica, foi utilizado um instrumento virtual utilizando a tecnologia VST. A tecnologia de estúdio virtual VST[15] é uma interface desenvolvida pela fabricante Steinberg em 1996 que integra sintetizadores e efeitos de áudio com editores e dispositivos de gravação de som. O VST utiliza processamento de sinal para simular o *hardware* desejado

[14] Disponível em <http://www.asio4all.com/>
[15] Abreviação do inglês *Virtual Studio Technology*

Figura 48: Configuração da interface MIDI para USB e acesso à placa de som do computador pelo ASIO4ALL.



Fonte: Elaborado pelo autor.

virtualmente. A sigla VSTi é normalmente usada para se referir a um instrumento virtual.

Os instrumentos virtuais VSTi são executados dentro de uma DAW, fazendo a ligação entre uma trilha gravada em MIDI com a execução dos áudios respectivos às notas MIDI. Em outras palavras, o VSTi lê um arquivo MIDI que não contém áudio e realiza a execução de sons a partir das notas lidas.

O VSTi utilizado dentro da DAW Nuendo foi adquirido gratuitamente através do repositório GTG[16]. O modelo utilizado foi o GTG DPC 3, que é um simulador de uma bateria acústica.

Quando o Nuendo recebe algum dado MIDI ele se baseia em um mapa de notas para escrever os dados na trilha de gravação. A partir do valor recebido via MIDI, o Nuendo procura a nota correspondente ao valor recebido em uma tabela que possui valores de 0 até 127 e escreve a nota na trilha de gravação. É importante não confundir o valor recebido via MIDI e o valor da nota escrito na gravação. Por exemplo, o Nuendo recebe o valor 36 via MIDI e então procura a nota correspondente a este valor para escrevê-la, porém é possível trocar a nota vinculada ao valor 36 a qualquer momento. Na Figura 49 podemos

[16] Disponível em <http://www.gtgsynths.com/plugins.htm>. Acesso em out. 2015.

Tabela 3: Mensagens MIDI utilizadas.

Status Byte	*Data Byte* 1	*Data Byte* 2	Descrição
1000nnnn	0kkkkkkk	0vvvvvvv	O *Status Byte* tem o comando para **desligar** a nota (*Note Off event*). Essa mensagem é enviada para indicar que uma nota deve ser desligada. O *Data Byte* 1 é o valor respectivo à nota do instrumento e o *Data Byte* 2 é a intensidade da nota.
1001nnnn	0kkkkkkk	0vvvvvvv	O *Status Byte* tem o comando para **ligar** a nota (*Note Off event*). Essa mensagem é enviada para indicar que uma nota deve ser ligada. O *Data Byte* 1 é o valor respectivo à nota do instrumento e o *Data Byte* 2 é a intensidade da nota.

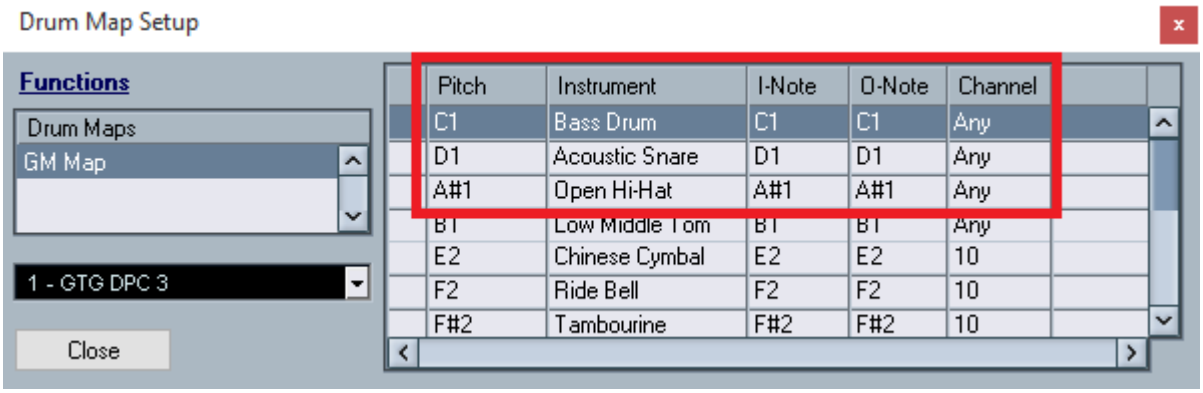
Nota 1: Os bits nnnn do *Status Byte* se referem aos 16 canais MIDI.

Nota 2: Os bits kkkkkkk do *Data Byte* 1 se referem à nota do instrumento.

Nota 3: Os bits vvvvvvv do *Data Byte* 2 se referem à intensidade da nota do *Data Byte* 1.

identificar que para o valor 36 recebido via MIDI a nota configurada é C1 (bumbo), para o valor 40 a nota configurada é D1 (caixa) e para o valor 46 a nota configurada é A#1 (chimbau).

Figura 49: Mapa de notas MIDI do Nuendo.



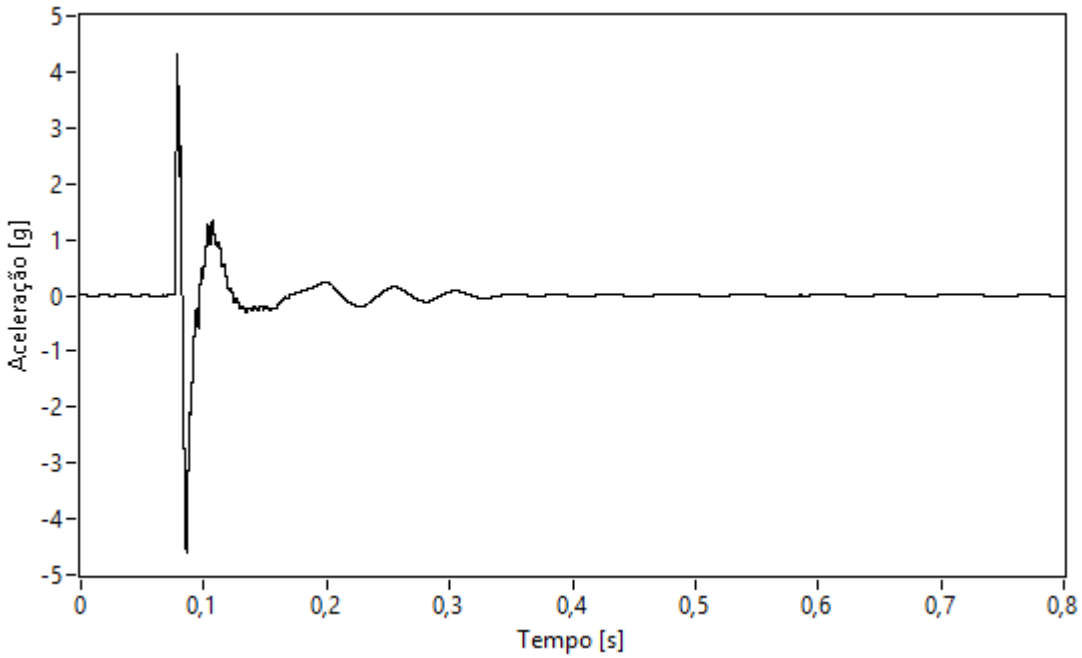
Fonte: Elaborado pelo autor.

4 RESULTADOS E DISCUSSÕES

4.1 Ensaios do Transdutor Piezoelétrico

Na Figura 50 encontra-se o sinal obtido do acelerômetro para somente um impacto e sua FFT na Figura 51. Na Figura 52 e na Figura 53 encontram-se o sinal obtido do piezoelétrico e sua FFT para um único impacto. A fim de melhor visualizar a comparação dos sinais dos dois sensores, na Figura 54 e na Figura 55 os sinais foram sobrepostos. Assim, é possível verificar claramente que a forma de onda do sinal do acelerômetro e do sinal do piezoelétrico são bastante semelhantes, bem como as FFT's.

Figura 50: Sinal obtido do acelerômetro para um impacto.

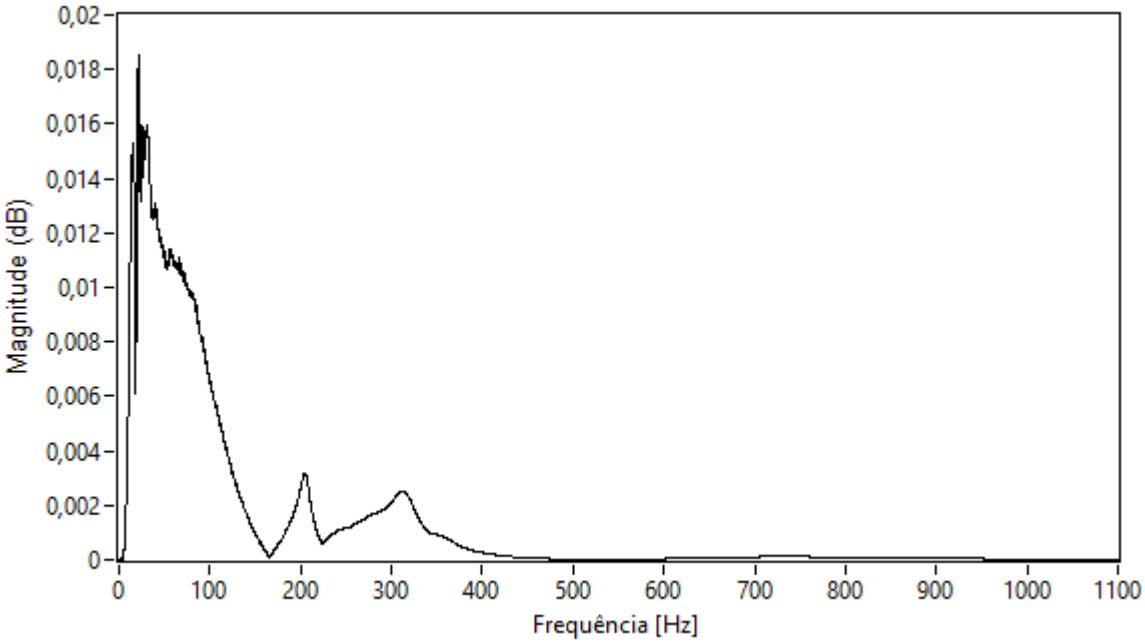


Fonte: Elaborado pelo autor.

Na Figura 56 encontra-se o sinal obtido do acelerômetro para varios impactos e sua FFT na Figura 57. Na Figura 58 e na Figura 59 encontram-se o sinal obtido do piezoelétrico e sua FFT para vários impactos. Da mesma forma feita anteriormente, na Figura 60 e na Figura 61 os sinais foram sobrepostos. Verifica-se novamente que a forma de onda do sinal do acelerômetro e do sinal do piezoelétrico são bastante semelhantes, bem como as FFT's.

É interessante notar na Figura 62 e na Figura 63, que são as FFT's aproximadas no gráfico, que os dois sinais apresentam um pico bem destacado na frequência de 16Hz,

Figura 51: FFT do sinal obtido do acelerômetro para um impacto.

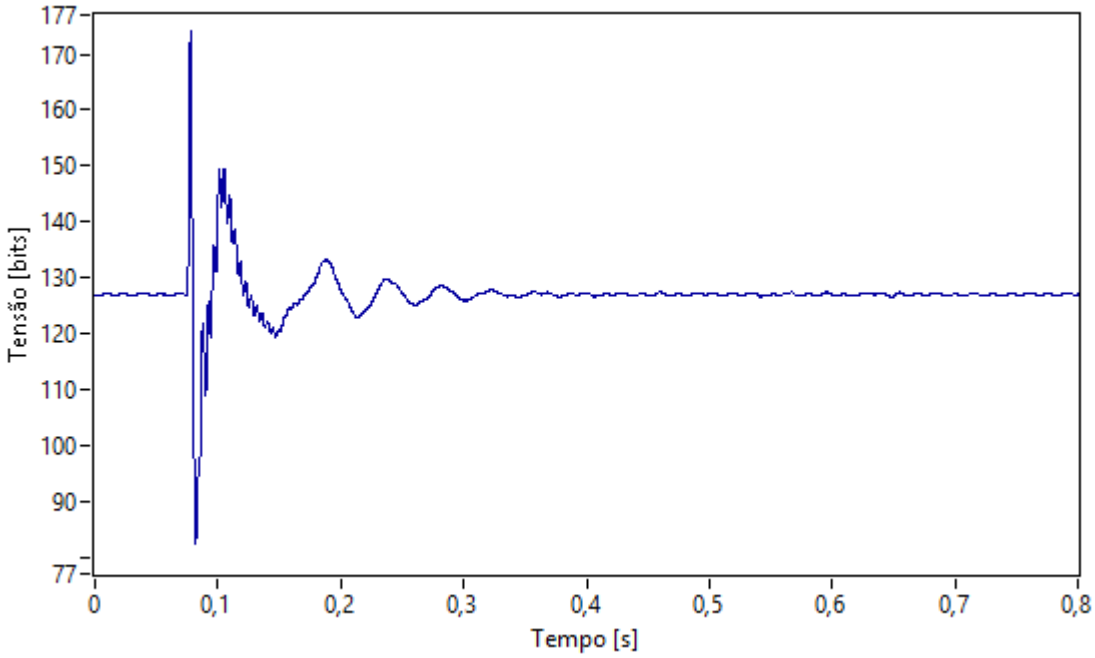


Fonte: Elaborado pelo autor.

que é a frequência dos impactos realizados.

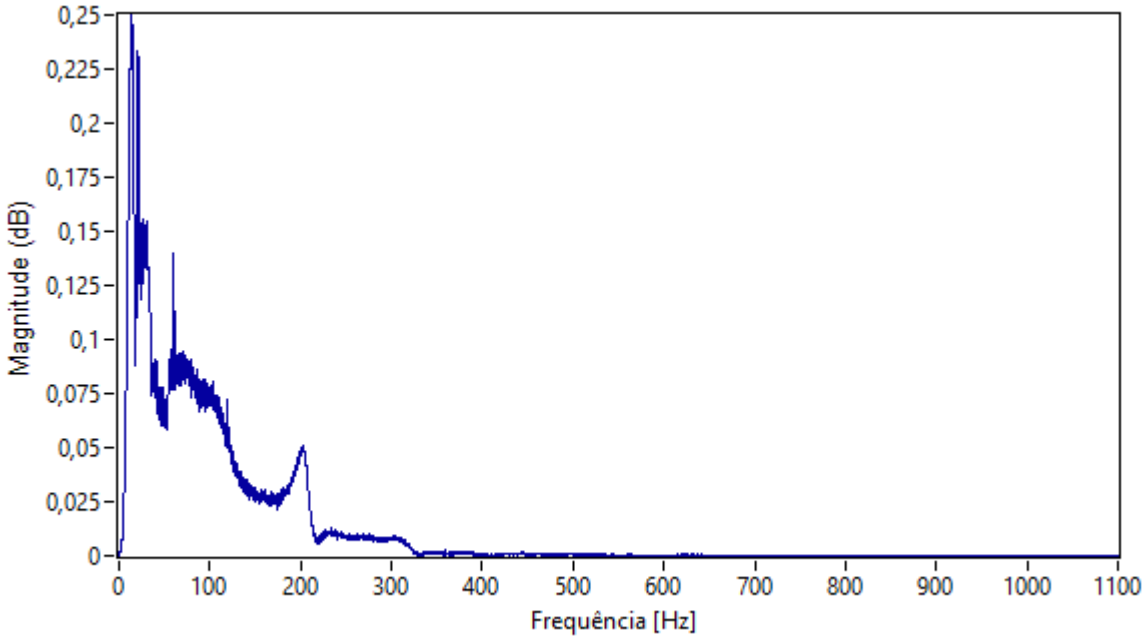
A partir dos resultados apresentados nesta seção, conclui-se que o sensor piezoelétrico se mostrou válido para utilização dentro da faixa de frequências desejada. Por outro lado, devido ao problema da não obtenção de informações do piezoelétrico, a modelagem elétrica utilizada está desconsiderando as constantes piezoelétricas do cristal, não sendo um modelo adequado para ser utilizado. Uma possibilidade mais adequada para o projeto seria a utilização de acelerômetros de baixo custo no lugar dos sensores piezoelétricos, assim podendo ter mais controle sobre a aquisição da vibração e mais confiança nos resultados obtidos. Vale lembrar que estes testes experimentais tiveram o objetivo apenas de verificar se o sensor poderia ser utilizado na faixa de frequências desejada e no modelo proposto, não entrando no escopo de calibração e frequência de ressonância do sensor.

Figura 52: Sinal obtido do piezoelétrico para um impacto.



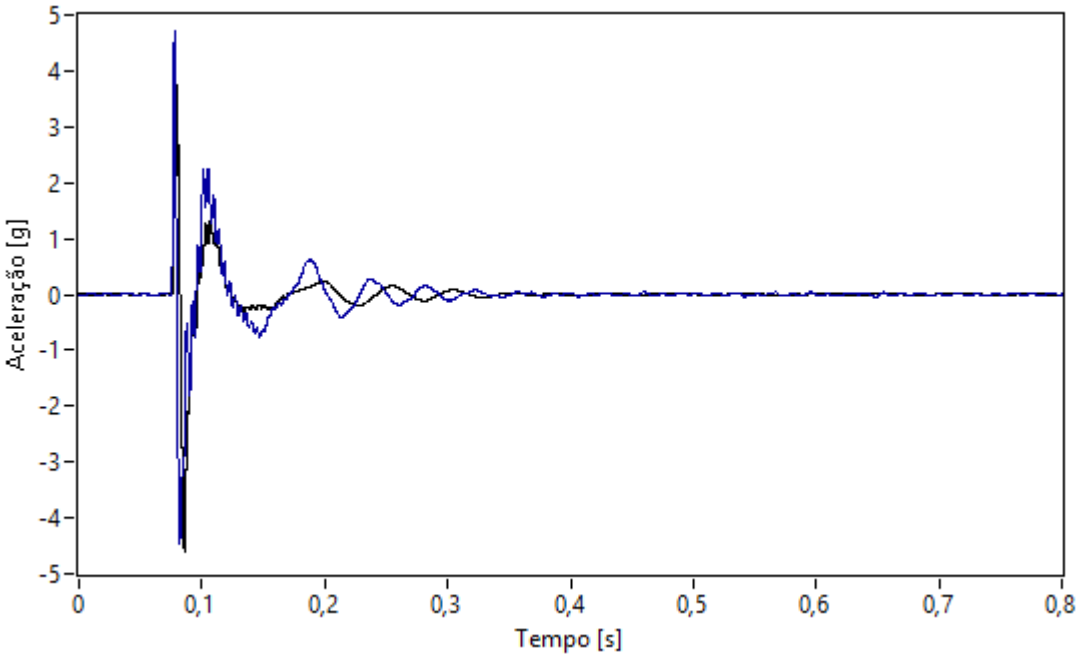
Fonte: Elaborado pelo autor.

Figura 53: FFT do sinal obtido do piezoelétrico para um impacto.



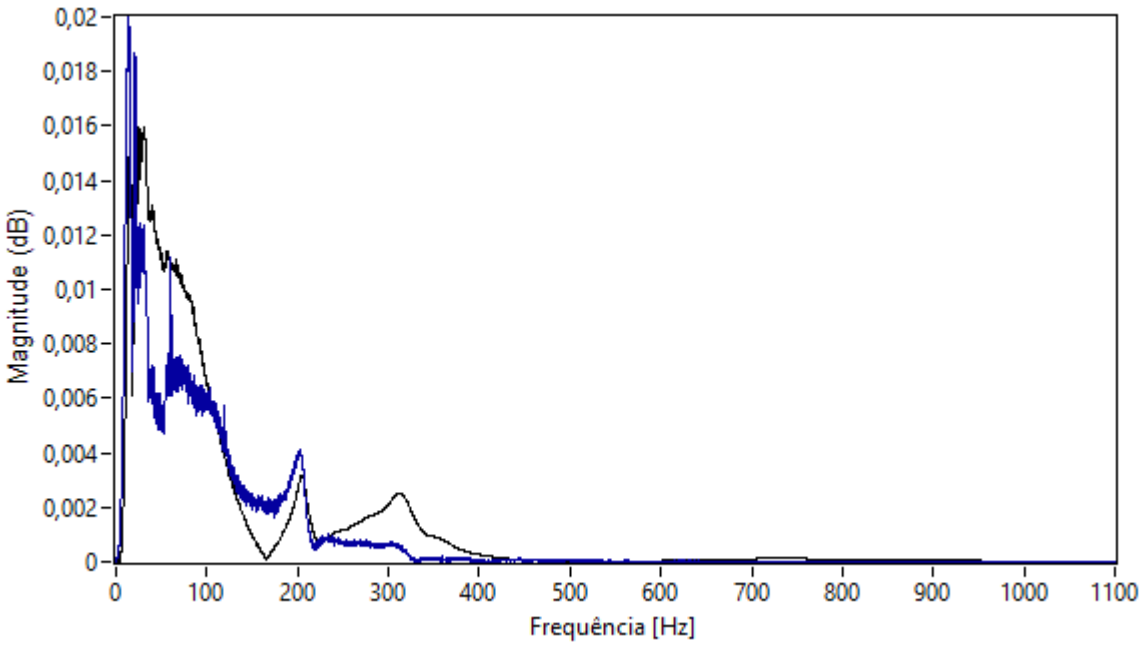
Fonte: Elaborado pelo autor.

Figura 54: Sinal obtido do acelerômetro e do piezoelétrico superpostos.



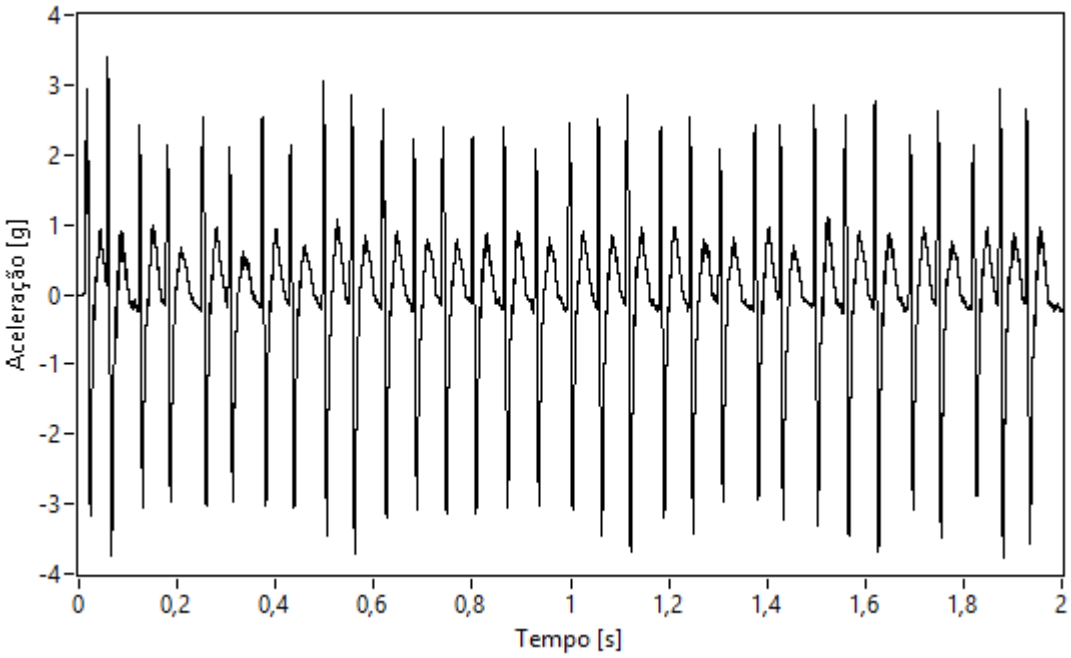
Fonte: Elaborado pelo autor.

Figura 55: FFT do sinal obtido do acelerômetro e do piezoelétrico superpostos.



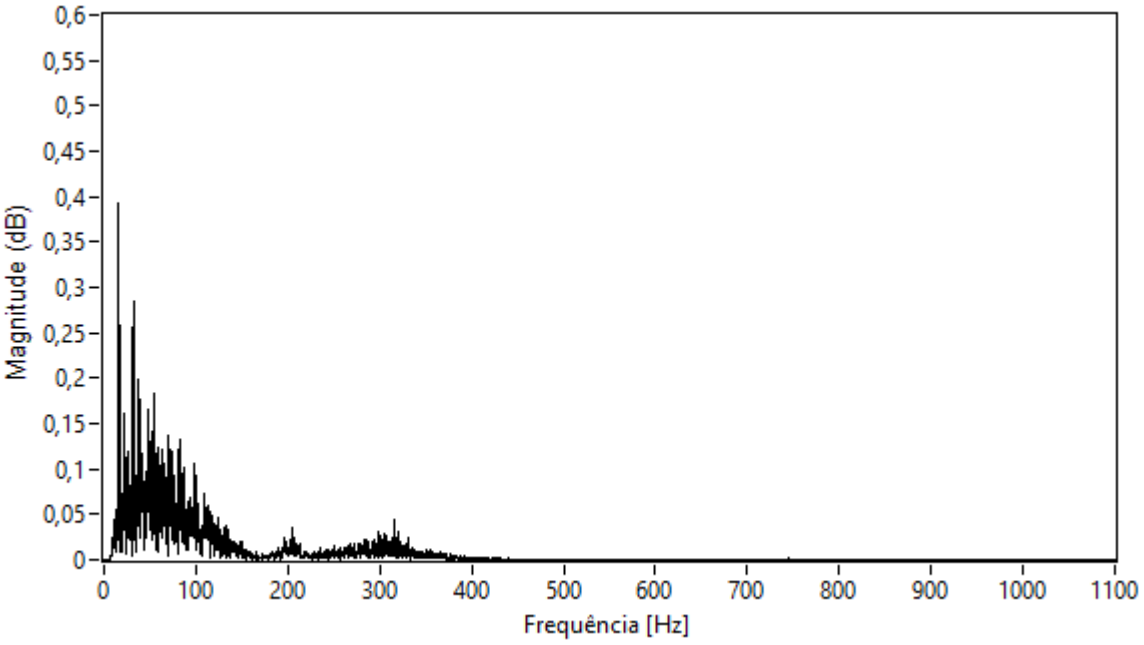
Fonte: Elaborado pelo autor.

Figura 56: Sinal obtido do acelerômetro para vários impactos.



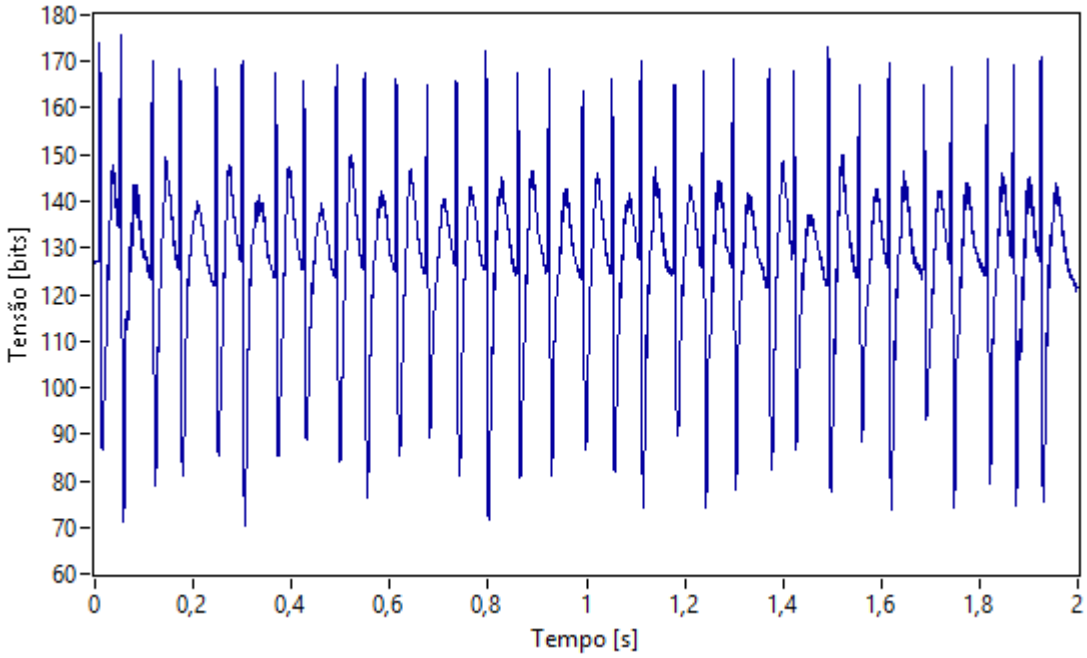
Fonte: Elaborado pelo autor.

Figura 57: FFT do sinal obtido do acelerômetro para vários impactos.



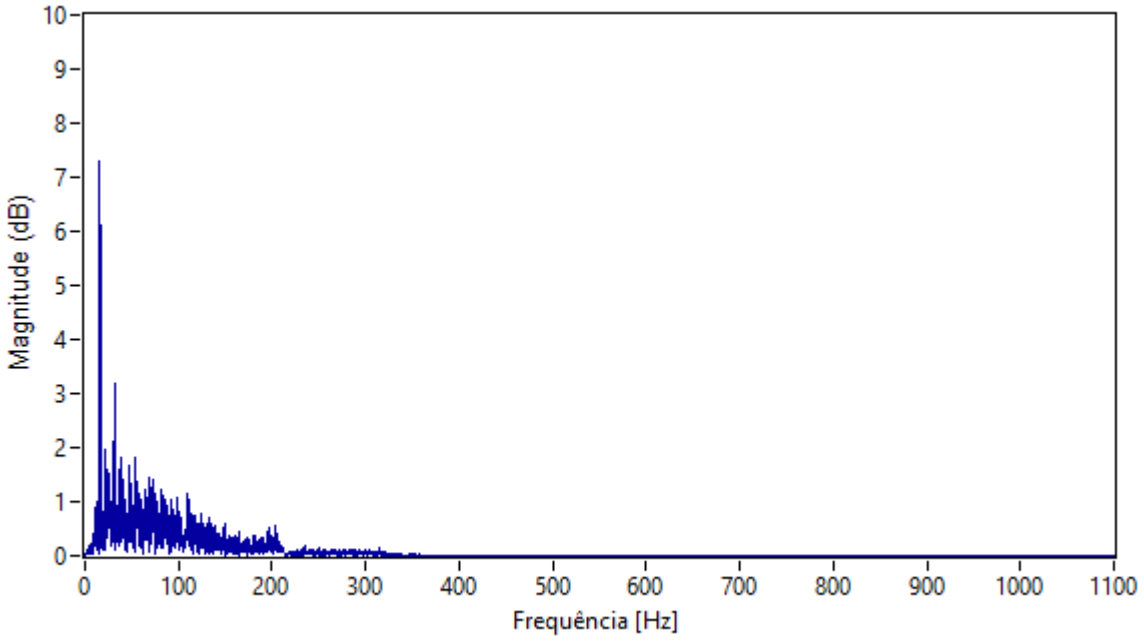
Fonte: Elaborado pelo autor.

Figura 58: Sinal obtido do piezoelétrico para vários impactos.



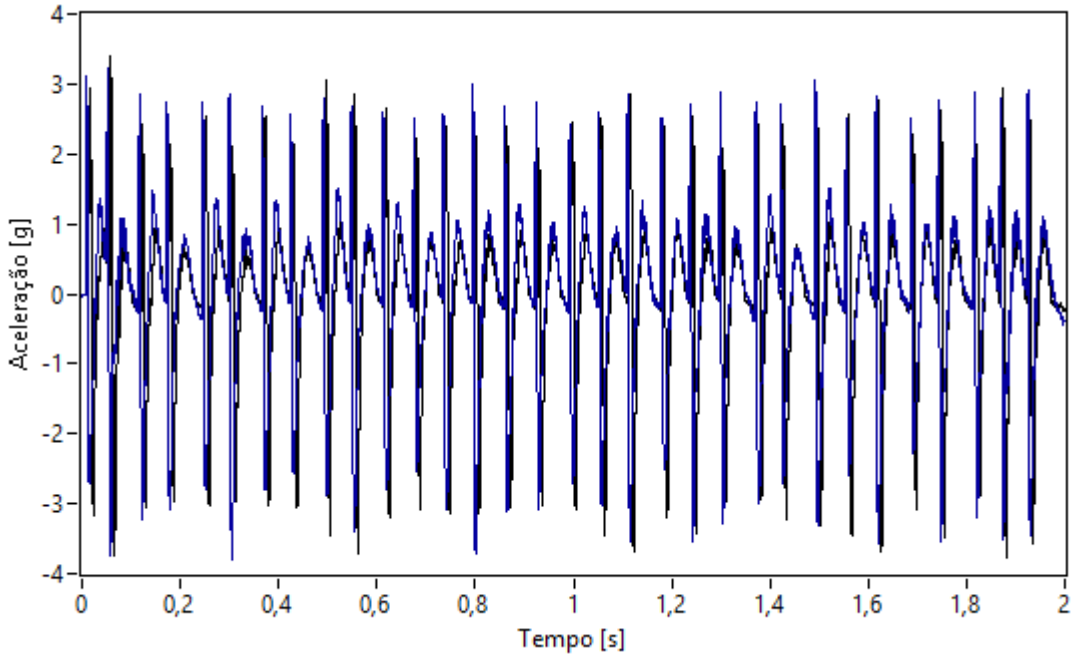
Fonte: Elaborado pelo autor.

Figura 59: FFT do sinal obtido do piezoelétrico para vários impactos.



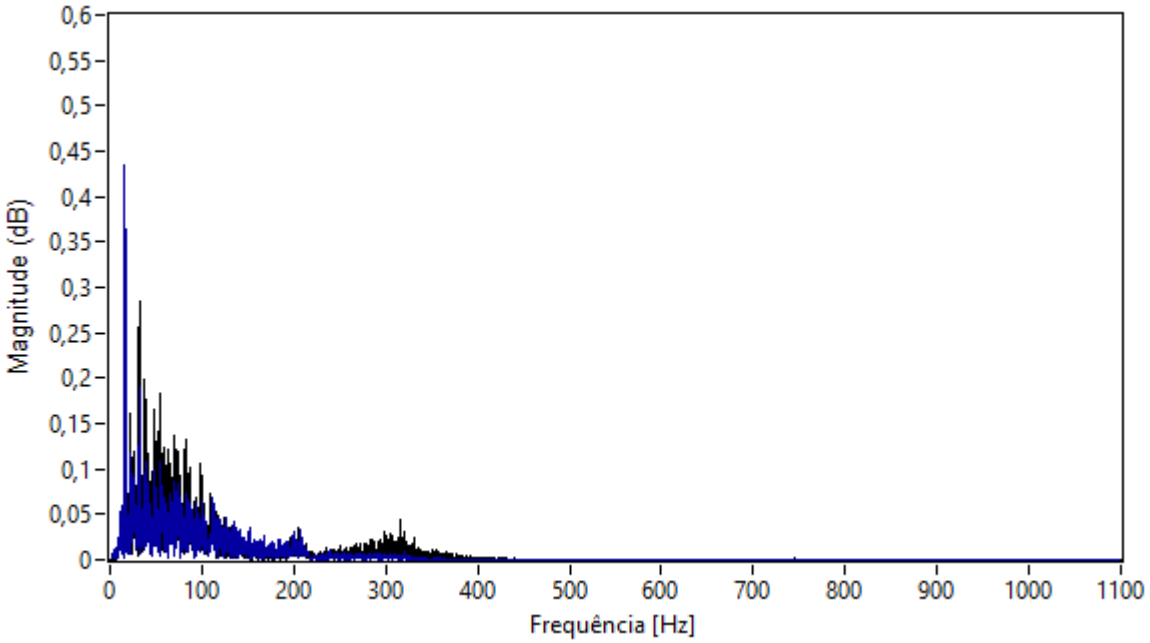
Fonte: Elaborado pelo autor.

Figura 60: Sinal obtido do acelerômetro e do piezoelétrico para vários impactos superpostos.



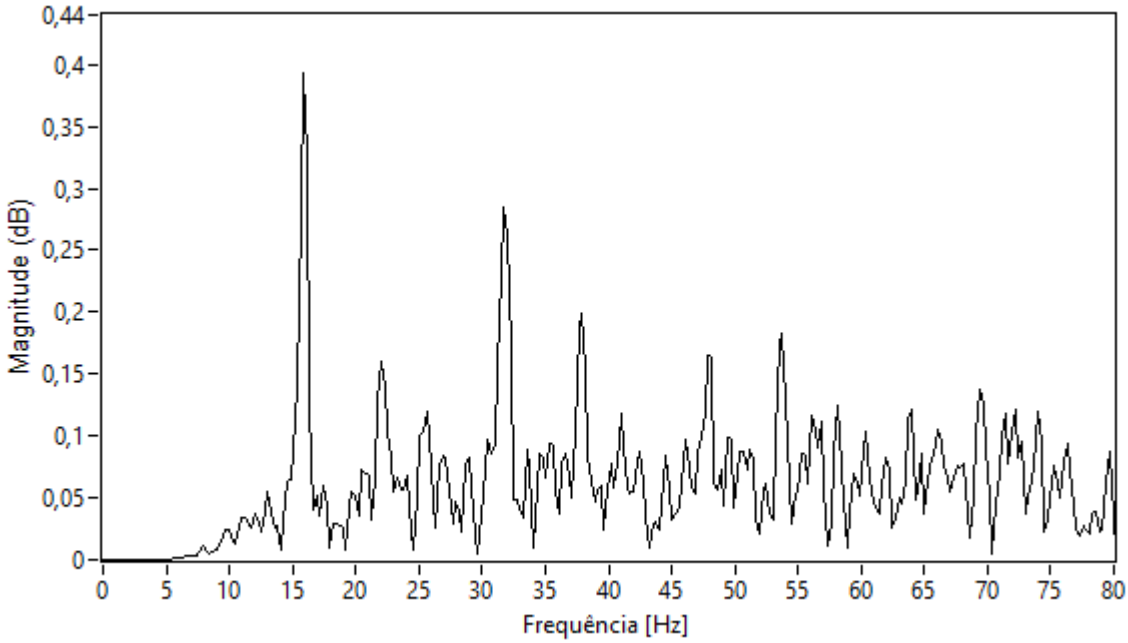
Fonte: Elaborado pelo autor.

Figura 61: FFT do sinal obtido do acelerômetro e do piezoelétrico para vários impactos superpostos.



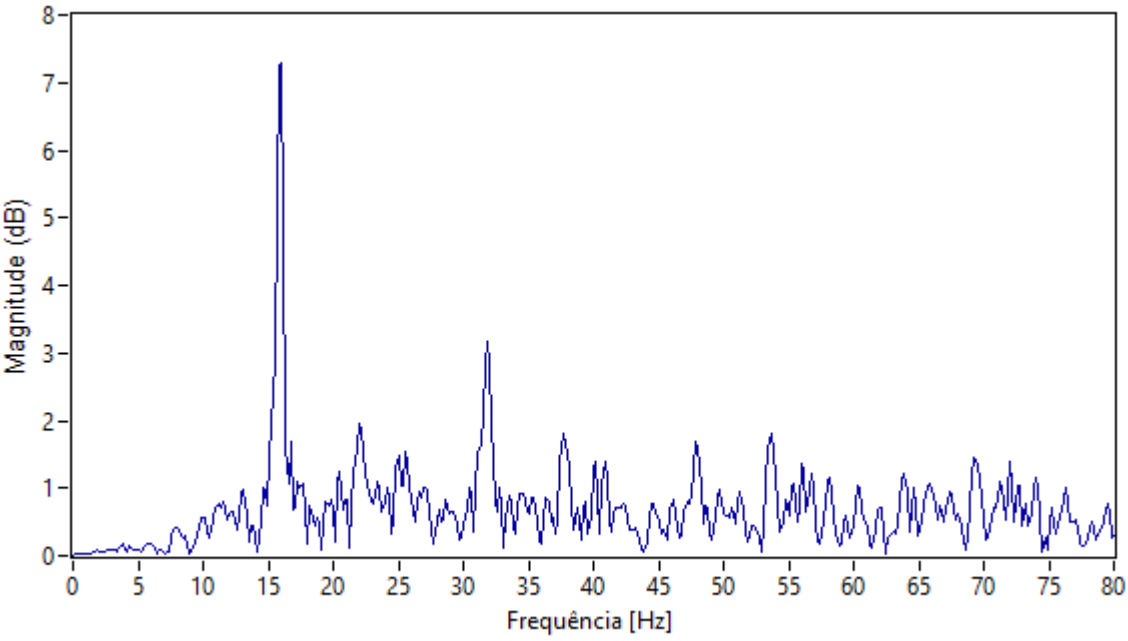
Fonte: Elaborado pelo autor.

Figura 62: Aproximação da FFT do sinal do acelerômetro para vários impactos.



Fonte: Elaborado pelo autor.

Figura 63: Aproximação da FFT do sinal do piezoelétrico para vários impactos.

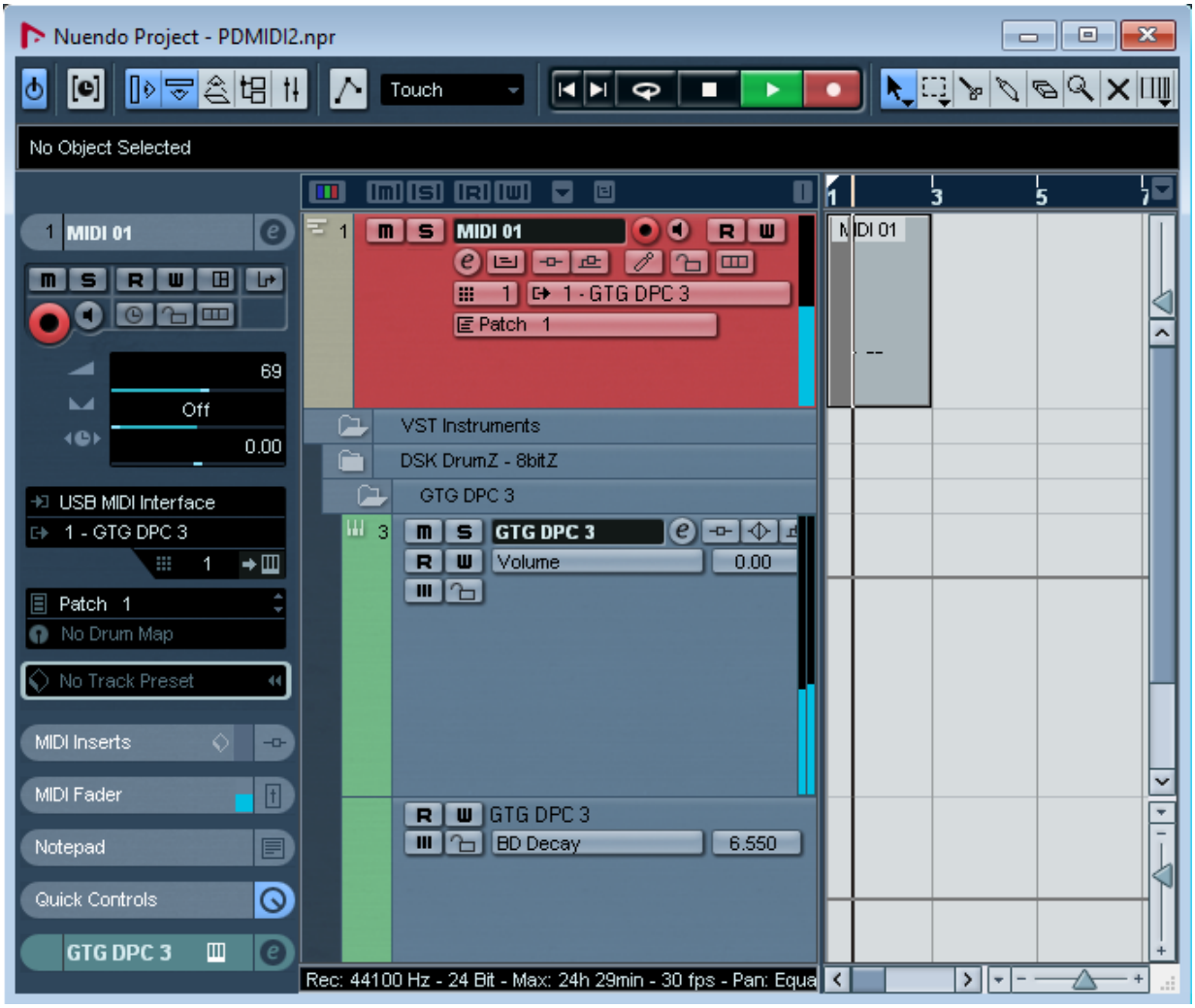


Fonte: Elaborado pelo autor.

4.2 Gravação da Execução por MIDI

Para verificar o funcionamento da gravação da execução do baterista, foi realizada um sequência de impactos nas três peças enquanto uma gravação estava em andamento no DAW Nuendo. A Figura 64 apresenta a interface do Nuendo enquanto a gravação estava ocorrendo, sendo possível visualizar a trilha sendo gravada na cor vermelha e ao lado uma barra azul indicando a intensidade da nota recebida naquele momento.

Figura 64: Interface do Nuendo quando ocorria a gravação da execução das notas.



Fonte: Elaborado pelo autor.

A Figura 65 mostra a sequência de notas recebidas e armazenadas após a gravação. Pode-se identificar o *status bytes* e os *data bytes* das notas executadas. Os valores do *data byte* 1 aparecem codificados pelo mapa de notas MIDI apresentado na Figura 49, sendo possível notar que as peças recebidas na sequência de execução são `C1 D1 A#1 C1 A#1 D1`, sendo C1 o bumbo, D1 a caixa e A#1 o chimbal.

Figura 65: Sequência das notas recebidas e gravadas.

Fonte: Elaborado pelo autor.

4.3 Execução do Áudio

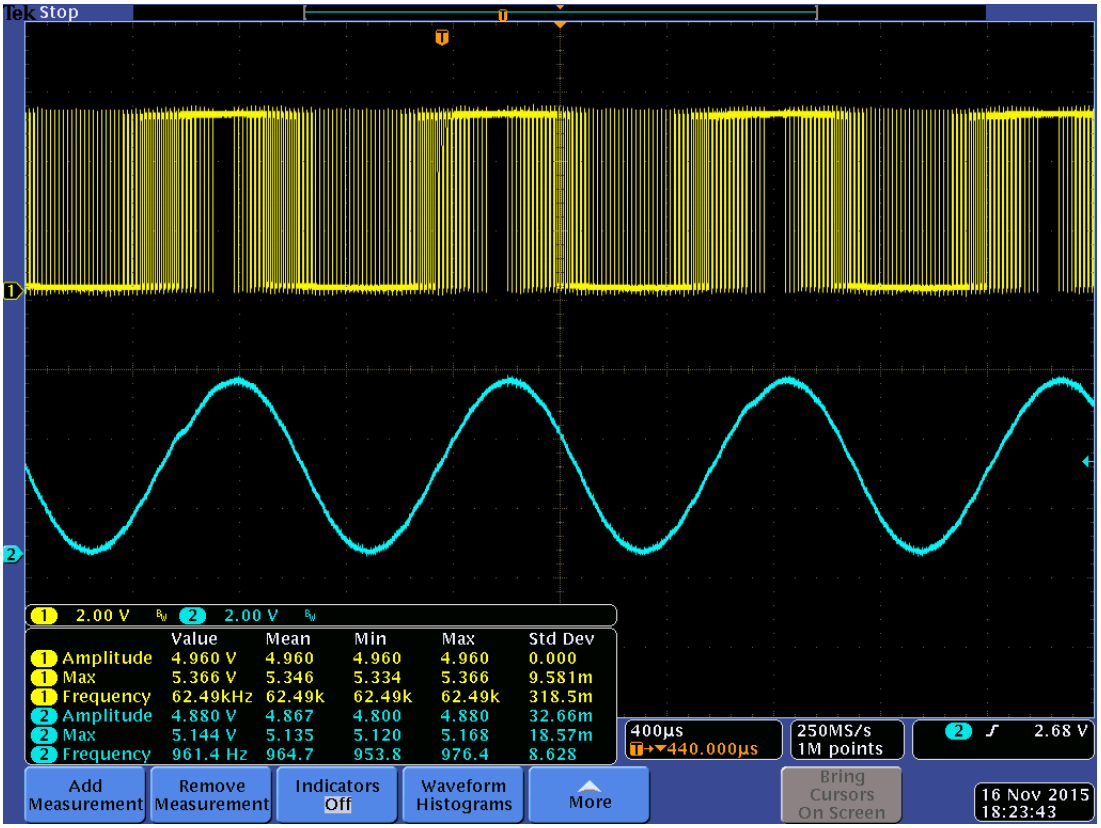
Nesta seção estão apresentados os resultados da execução do áudio na saída PWM do ATmega2560. Todas as imagens foram adquiridas através do osciloscópio de modelo MSO4000B da fabricante Tektronix.

Como primeira análise do sinal PWM e da saída do filtro de demodulação, foi carregado no ATmega2560, através de uma *lookup table*, um sinal senoidal amostrado em 1kHz e quantizado em 8 *bits*. A *lookup table* foi carregada com os valores *128,176,218,245,255,245,218,176,128,79,37,10,0,10,37,79*, sendo esses valores criados com o auxílio de uma ferramenta disponível gratuitamente na Internet no sítio digital <http://www.daycounter.com/Calculators/Sine-Generator-Calculator.phtml>.

A tabela possui 16 valores distintos respectivos à amostras de um seno de amplitude de 8 *bits* e, como a frequência de atualização do *duty cycle* do PWM é de 16kHz, obteve-se a saída PWM correspondente à um seno de 1kHz ($16kHz/16 = 1kHz$). A saída PWM correspondente aos valores da tabela do seno está apresentada na Figura 66 no canal de cor amarela e a saída do filtro de demodulação está apresentada no canal de cor azul. As medições de amplitude e frequência dos canais são apresentadas da tela do osciloscópio. Pode-se identificar que a frequência medida do PWM é 62,49kHz, confirmando o valor configurado no ATmega2560, e a frequência medida do sinal senoidal obtido na saída do filtro é 961,4Hz. As amplitudes observadas possuem valor de 4,980V para o PWM e 4,880V para o sinal após o filtro. A análise com a geração de um sinal senoidal através do PWM demonstrou desempenho condizente ao estudo do filtro de demodulação proposto

anteriormente, obtendo-se diferença de amplitude de 110mV e atraso de 52μs.

Figura 66: Saída PWM do ATmega2560 e saída do filtro de demodulação.

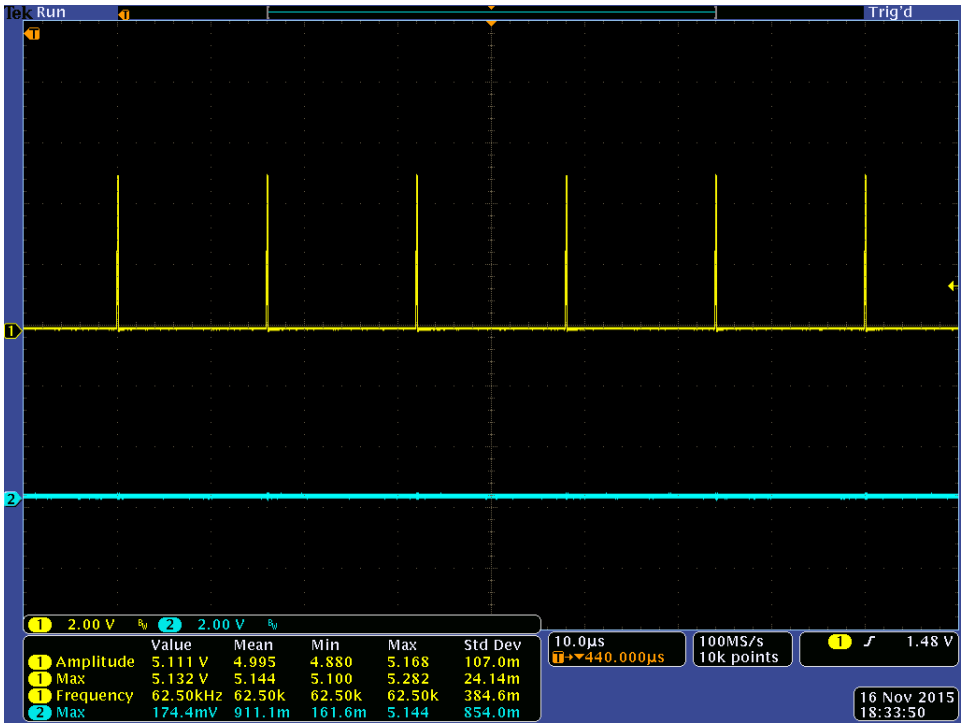


Fonte: Elaborado pelo autor.

A segunda análise foi realizada com sinais PWM de *duty cycle* constantes. Os valores para o *duty cycle* utilizados foram 1%, 50% e 99%. A Figura 67 apresenta o resultado do filtro com *duty cycle* de 1%, a tensão DC da saída do filtro medida no osciloscópio é 174,4mV e a tensão de pico do PWM foi de 5,132V. Já a Figura 68 apresenta saída do filtro com 2,640V e por fim a Figura 69 apresenta saída do filtro de 5,116V.

A terceira e última análise foi realizada com um dos áudios gravados no cartão SD. O arquivo escolhido para reprodução foi o "bass16k.wav" pois é o áudio de menor duração, facilitando a análise. Na Figura 70 é possível visualizar o sinal sendo gerado através do PWM e a saída do filtro de demodulação. Ao analisar o sinal demodulado do áudio e compará-lo ao sinal de áudio aberto no *software* Audacity na Figura 71, é possível perceber que as formas de onda são bastante similares, comprovando a fidelidade na execução do áudio gerado pelo PWM do ATmega2560 e demodulado pelo filtro projetado. Além disso, sabendo que o áudio executado possui duração de 85ms através do *software* Audacity, percebe-se que o sinal demodulado possui a mesma duração do áudio, visto que o sinal termina em 8,5 divisões de 10ms cada no osciloscópio.

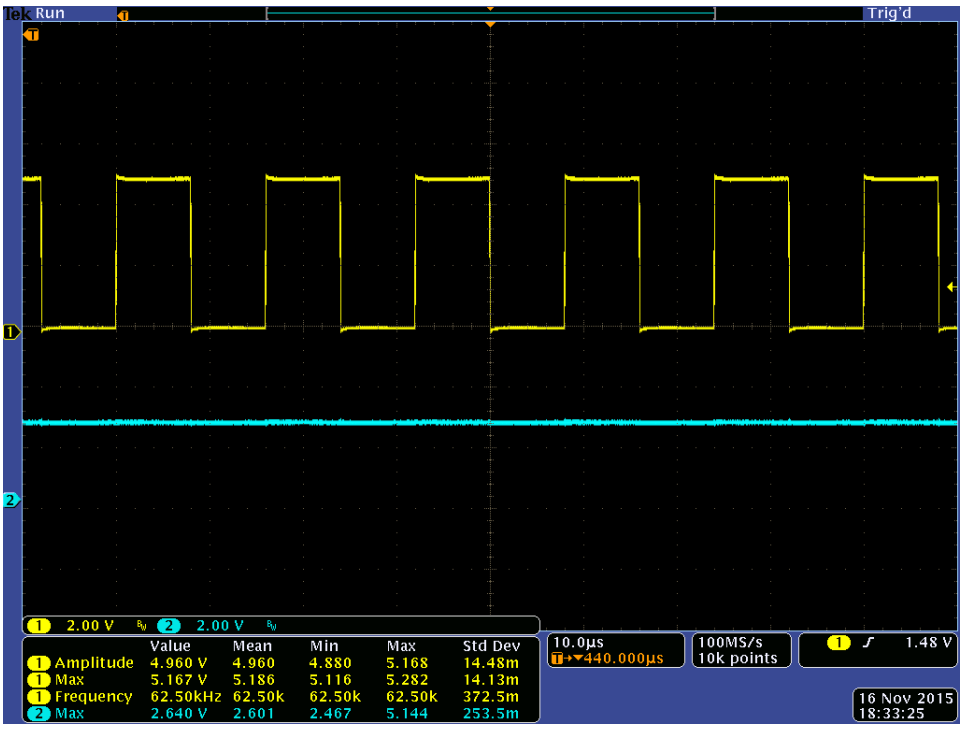
Figura 67: Saída PWM do ATmega2560 com *duty cycle* de 1% e saída do filtro de demodulação com tensão de 174,4mV.



Fonte: Elaborado pelo autor.

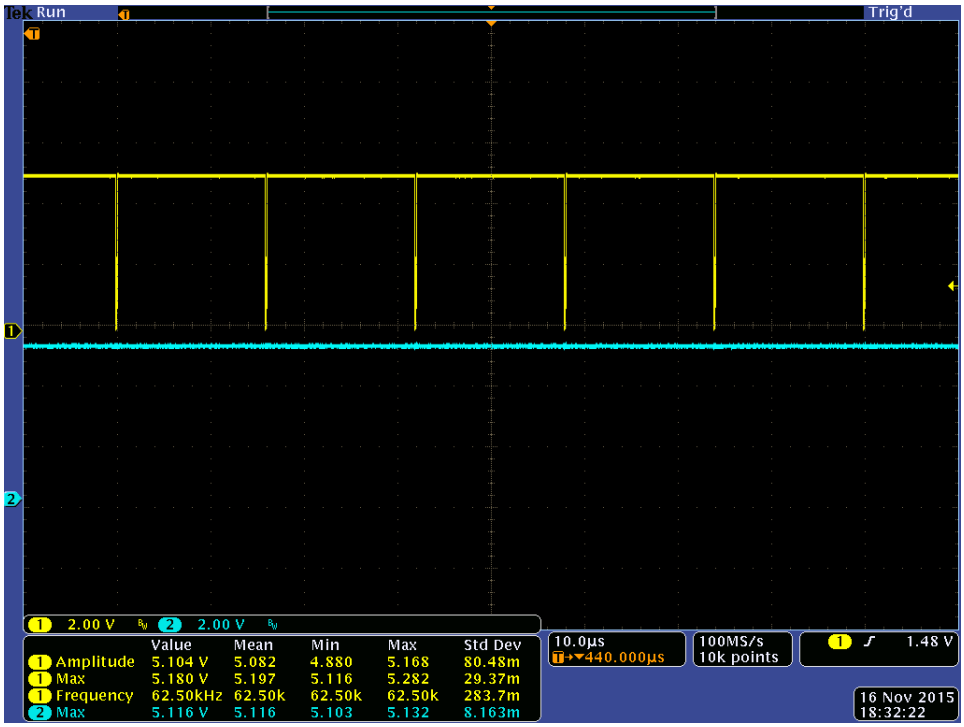
Na Figura 72 é possível identificar com mais detalhes o sinal de áudio gerado e demodulado. No sinal PWM nota-se claramente que cada *byte* lido do arquivo de áudio dura quase 4 períodos do PWM. Como a frequência do PWM é 62,5kHz e a frequência do áudio é 16kHz, dividindo-se 62,5kHz por 16kHz encontra-se que a frequência do áudio é de 3,9 vezes a frequência do PWM, indicando que cada *byte* do arquivo de áudio será executado por 3,9 períodos do PWM.

Figura 68: Saída PWM do ATmega2560 com *duty cycle* de 50% e saída do filtro de demodulação com tensão de 2,640V.



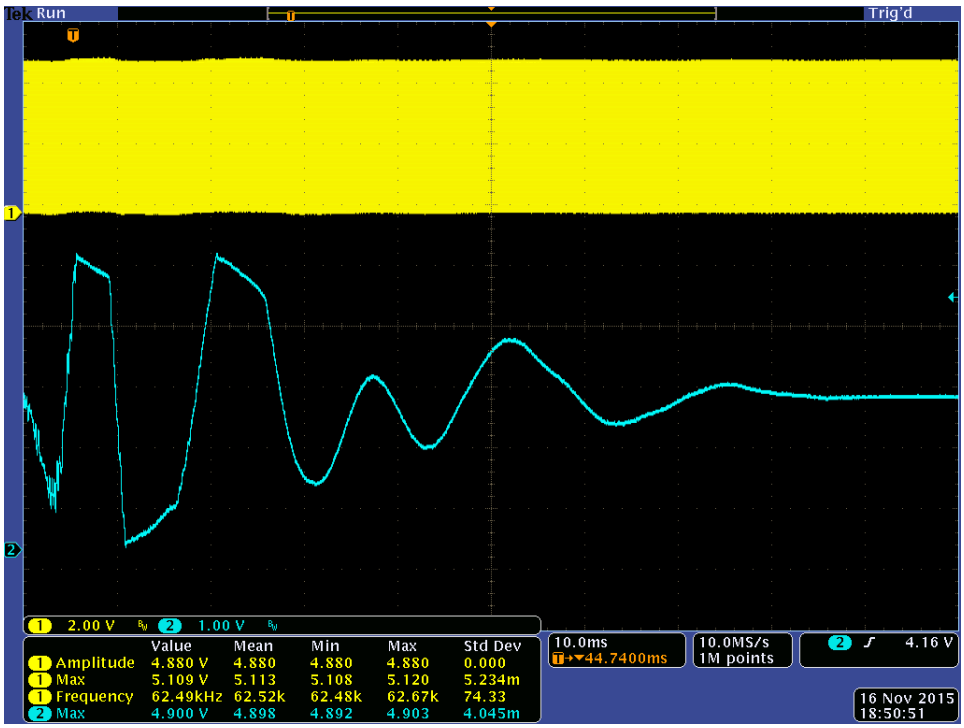
Fonte: Elaborado pelo autor.

Figura 69: Saída PWM do ATmega2560 com *duty cycle* de 99% e saída do filtro de demodulação com tensão de 5,116V.



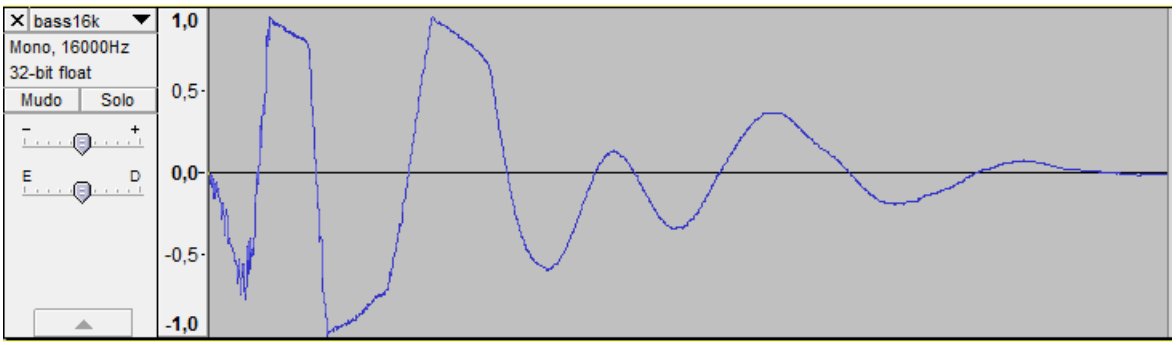
Fonte: Elaborado pelo autor.

Figura 70: Saída PWM da execução do áudio "bass16k.wav" armazenado no cartão SD.



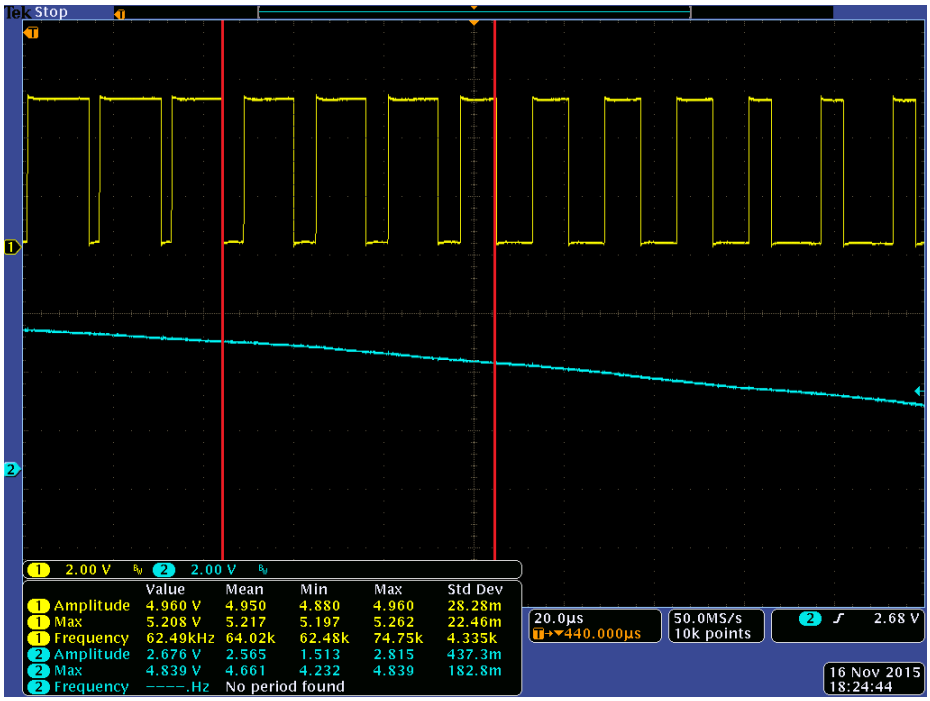
Fonte: Elaborado pelo autor.

Figura 71: Arquivo de áudio “bass16k.wav” aberto no *software* Audacity.



Fonte: Elaborado pelo autor.

Figura 72: Saída PWM da execução do áudio “bass16k.wav” armazenado no cartão SD em detalhes.



Fonte: Elaborado pelo autor.

5 CONCLUSÕES

Nos experimentos realizados, o sistema conseguiu simular uma bateria eletrônica sem fio, cumprindo assim sua função principal. A proposta da comunicação sem fio entre as peças da bateria e o módulo de processamento central foi realizada obtendo-se resultados aceitáveis, onde a latência na transmissão dos sinais não é perceptível para um músico. Vale lembrar que um equipamento similar não existe no mercado sendo possível futuramente a continuação e melhoria deste projeto a um nível comercial.

A utilização do transdutor piezoelétrico se mostrou válida nos experimentos para aquisição do impacto realizado pelo usuário em uma das peças da bateria, com alto custo-benefício. Porém, não se pode ter controle da parte de instrumentação do piezoelétrico pois não foram conhecidos as constantes piezoelétricas do cristal utilizado, o que impossibilitou uma modelagem elétrica e um condicionamento adequado. Também não é conhecida a faixa de frequência do cristal piezoelétrico utilizado, não podendo ser possível ter certeza do funcionamento nas frequências projetadas para o condicionador. Uma possibilidade a ser investigada em um projeto futuro seria a utilização de acelerômetros de baixo custo no lugar dos sensores piezoelétricos, assim podendo ter mais controle sobre a aquisição da vibração e mais confiança nos resultados obtidos.

A proposta de implementação de um menu para o usuário realizar algumas configurações pertinentes não foi implementada. Outras funcionalidades propostas, tais como a execução do áudio armazenado em um cartão SD e a gravação da execução do baterista em um *software* de edição de áudio foram implementadas com sucesso.

Apesar das limitações enfrentadas na parte da instrumentação do piezoelétrico e da falta do menu de configuração, o protótipo simulou as funcionalidades básicas para uma bateria eletrônica sem fio, captando as amplitudes dos impactos realizados pelo usuário que são transmitidas sem fio e realizando o processamento proposto para os microcontroladores corretamente. Com isso, podemos considerar o projeto desenvolvido válido para a aplicação desejada. Com um maior período de tempo para desenvolvimento do projeto certamente os objetivos seriam expandidos e um produto final com uma maior qualidade e quantidade de funções teria sido implementado.

O trabalho desenvolvido mostrou-se um projeto interdisciplinar no qual foi empregado conhecimentos de diferentes disciplinas do curso de Engenharia Elétrica, tais como microcontroladores, instrumentação e eletrônica. Desta forma o projeto desenvolvido pode ser visto não somente como uma forma de preparação para o mercado de trabalho mas também como um processo importantíssimo na formação do aluno, uma vez que foi necessária a aplicação e interconexão de diversas áreas da Engenharia Elétrica.

6 PROPOSTA PARA TRABALHOS FUTUROS

Estudos mais aprofundados do sensor piezoelétrico a fim de possibilitar seu tratamento e utilização adequado são fundamentais para uma melhoria do sistema. Uma substituição dos sensores utilizados por acelerômetros de baixo custo é uma boa alternativa às limitações enfrentadas na utilização do piezoelétrico. Seria interessante realizar uma análise comparativa entre a utilização de acelerômetros de baixo custo e de sensores piezoelétricos, definindo as vantagens e desvantagens de cada um relacionadas ao custo e benefícios.

A realização da medição do valor de latência entre o impacto realizado e a execução do som tornaria a análise dos atrasos na transmissão sem fio mais precisas. Assim, teria-se mais controle na otimização da transmissão sem fio, não sendo necessária a análise por um músico experiente.

Uma interface gráfica para configuração de parâmetros do protótipo pelo usuário tornaria o equipamento mais funcional e interativo, fornecendo ao usuário maior controle do seu equipamento.

O protótipo ficaria mais funcional se fosse implementada rotinas no *firmware* que permitissem executar mais tipos de arquivos WAVE, com diferentes taxas de amostragem, quantidade de canais e níveis de quantização. Com isso o usuário não precisaria se preocupar em tratar o arquivo de áudio.

Na execução do som respectivo à peça percutida, poderia ser utilizado um conversor digital-analógico no lugar do PWM e do filtro, garantindo maior qualidade no áudio gerado. Um controle de volume do áudio gerado, implementado digitalmente ou analogicamente, forneceria ainda mais controle ao usuário.

Poderia ser implementada uma quantidade maior de mensagens MIDI possíveis de serem enviadas, proporcionando uma melhor experiência na gravação da execução do baterista. Além disso, uma investigação importante seria uma possível melhoria na eficiência dos componentes das peças a fim de diminuir o consumo de energia.

REFERÊNCIAS

ABEL, A. M. da S.; LUIZ, S. G.; SHIGUE, C. Sensores e atuadores piezoelétricos. Escola de Engenharia de Lorena, São Paulo, 2010. Citado na página 27.

ADAMOWSKI, J. C. *Sensores: Teoria e Aplicações*. 3. ed. São Paulo: Escola Politécnica da USP, 2000. Citado na página 27.

ALLSYLLABUS.COM. *Quantization Process*. 2015. Disponível em: <http://www.allsyllabus.com/aj/note/ECE/Digital%20Communication/unit3/Quantization%20Process.php#.VkaIz_mrTIV>. Acesso em: 13 nov. 2015. Citado na página 29.

ATMEL. *ATmega2560 Datasheet*. San Jose, CA, 2014. Disponível em: <http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf>. Acesso em: 06 nov. 2015. Citado 2 vezes nas páginas 42 e 44.

BADNESS, R. *Drum Programming: A Complete Guide to Program and Think Like a Drummer*. Anaheim Hills, CA: Ray Badness, 1991. Citado 2 vezes nas páginas 20 e 21.

BALBINOT, A.; BRUSAMARELLO, V. J. *Instrumentação e Fundamentos de Medidas - Vol.2*. 2. ed. Rio de Janeiro, RJ, 2011. Citado 2 vezes nas páginas 28 e 60.

COMITTEE, J. M. S. *MIDI 1.0 Detailed Specification*. 4.2. ed. Los Angeles, CA, 1995. Disponível em: <http://oktopus.hu/uploaded/Tudastar/MIDI%201.0%20Detailed%20Specification.pdf>. Acesso em: 19 nov. 2015. Citado 4 vezes nas páginas 22, 23, 24 e 25.

HANSEN, B. *Musical Instrument Digital Interface (MIDI)*. Portland, OR, 2005. Disponível em: <http://www.hansenb.pdx.edu/pdf/MIDI.pdf>. Acesso em: 19 nov. 2015. Citado 3 vezes nas páginas 22, 24 e 25.

HIGASHI, T. *LCR-745 Digital Meter Instruction Manual*. Yokohama, Japão, 1990. Citado na página 57.

IBM; MICROSOFT. *Multimedia Programming Interface and Data Specifications 1.0*. [S.l.], 1991. Disponível em: <http://www-mmsp.ece.mcgill.ca/documents/audioformats/wave/Docs/riffmci.pdf>. Acesso em: 25 out. 2015. Citado 2 vezes nas páginas 29 e 30.

INTERNATIONAL, D. *ZigBee RF Modules User Guide*. Revision w. Minnetonka, MN, 2015. Disponível em: <http://ftp1.digi.com/support/documentation/90000976_W.pdf>. Acesso em: 4 out. 2015. Citado na página 26.

KARKI, J. *Signal Conditioning Piezoelectric Sensors*. [S.l.], 2000. Disponível em: <http://www.ti.com/lit/an/sloa033a/sloa033a.pdf>. Acesso em: 17 nov. 2015. Citado 3 vezes nas páginas 27, 58 e 59.

KERR, D. *MIDI — The Musical Instrument Digital Interface*. Cleveland, OH, 2009. Disponível em: <http://dougkerr.net/Pumpkin/articles/MIDI.pdf>. Acesso em: 17 nov. 2015. Citado 4 vezes nas páginas 21, 23, 24 e 25.

MAXSTREAM, I. *Product Manual - XBee Series 2 OEM RF Modules*. 1.0.1. ed. Lindon, UT, 2015. Disponível em: <http://ftp1.digi.com/support/documentation/manual_xb_oem-rf-modules_802.15.4_v1.xAx.pdf>. Acesso em: 4 out. 2015. Citado na página 26.

MEASUREMENTSPECIALTIES. *Interfacing Piezo Film to Electronics*. Hampton, VA, 2006. Disponível em: <http://www.eng.hmc.edu/NewE80/PDFs/PiezoElectronics.pdf>. Acesso em: 13 nov. 2015. Citado na página 29.

MICROCHIP. *PIC18F2550 Data Sheet*. USA, 2006. Disponível em: <http://ww1.microchip.com/downloads/en/devicedoc/39632c.pdf>. Acesso em: 06 nov. 2015. Citado na página 47.

MICROSOFT. *New Multimedia Data Types and Data Techniques*. Revision 3.0. [S.l.], 1994. Disponível em: <http://www-mmsp.ece.mcgill.ca/documents/audioformats/wave/Docs/RIFFNEW.pdf>. Acesso em: 25 out. 2015. Citado 3 vezes nas páginas 28, 30 e 32.

SANDISK. *SanDisk Secure Digital Card Product Manual*. 1.9. ed. Sunnyvale, CA, 2003. Disponível em: <http://www.circlemud.org/jelson/sdcard/SDCardStandardv1.9.pdf>. Acesso em: 12 out. 2015. Citado na página 50.

SHACKELFORD, J. F. *Introdução à ciência dos materiais para engenheiros*. São Paulo: Pearson Prentice Hall, 2008. Citado na página 27.

SIMMONS. *SDS 1 Operating Instructions*. [S.l.], 1978. Disponível em: <http://www.simmonsmuseum.com/?area=downloads&download_id=121>. Acesso em: 19 nov. 2015. Citado na página 21.

APÊNDICE A – COMANDOS AT PARA CONFIGURAÇÃO DO *BAUD RATE* DO XBEE

No terminal do XCTU, com o XBee do coordenador conectado, foram enviados os seguintes comandos AT para a configuração de 31250 *baud*:

```
+++      //Comando para entrar no modo de configuração do XBee
ATBD7A12 //Comando para definir o baud rate. O valor
         //hexadecimal 7A12 corresponde ao valor em decimal de 31250
ATWR
ATAC
ATCN
```

APÊNDICE B – CÓDIGOS DO MÓDULOS DO ATMEGA2560

Arquivo “timer1.c”:

```
/*****************************************************************/
/*                     Modulo do Timer 1                         */
/*****************************************************************/
//Modulo responsavel por realizar um temporizacao de 1ms onde,
//na rotina de interrupcao, sao executadas diversas funcoes

//Inclusoes modulos necessarios
#include "main.h"
#include "timer1.h"
#include "lcd.h"

//Macros para temporizacoes especificas
#define TIMEOUT_STARTUP_SCREEN  1000
#define TIMEOUT_PAD_COUNTER             100

//Declaracao e inicializacao de variaveis do modulo
unsigned int contador_lcd = 0;
unsigned int pad1_counter = 0;
unsigned int pad2_counter = 0;
unsigned int pad3_counter = 0;

void TIMER1_init(void){

        // Timer/Counter 1 initialization
        // Clock source: System Clock
        // Clock value: 16000,000 kHz
        // Mode: Normal top=0xFFFF
        // Timer Period: 1 ms
        // Timer1 Overflow Interrupt: On
        TCCR1B = (1<<CS10);
        TCNT1H=0xC1;
        TCNT1L=0x80;

        // Timer/Counter 1 Interrupt(s) initialization
        TIMSK1 = (1<<TOIE1);
```

```
36
37          CLEAR_FLAG(STARTUP_SCREEN_DONE); //Limpa flag de
               temporizacao da tela inicial
38  }
39
40  //Rotina de interrupcao
41  ISR(TIMER1_OVF_vect){
42
43          GLOBAL_INTERRUPT_DISABLE;
44
45          //Recarrega os registradores do timer1
46          TCNT1H=0xC1;
47          TCNT1L=0x80;
48
49          //Rotina para a temporizacao da tela de inicio
50          if (contador_lcd < TIMEOUT_STARTUP_SCREEN) //Contador do
               tempo para a tela inicial ficar aparecendo
51          {
52                  contador_lcd++;
53          } else if (contador_lcd == TIMEOUT_STARTUP_SCREEN)
54          {
55                  SET_FLAG(STARTUP_SCREEN_DONE);
56                  contador_lcd++;
57          }
58
59          /*********************************************************/
60          /*    Rotinas para verificar se algum PAD foi tocado   */
61          /*********************************************************/
62          //Se algum PAD foi tocado, realiza uma temporizacao que e
63          //utilizada para o LCD e para o envio MIDI
64
65          if (READ_FLAG(PAD1_ON))
66          {
67                  pad1_counter++;
68                  if (pad1_counter == TIMEOUT_PAD_COUNTER)
69                  {
70                          pad1_counter = 0;
71                          CLEAR_FLAG(PAD1_ON);
72                          lcd_pad_state(1,OFF); //Para indicar no
                               LCD que o PAD esta desligado
73                  }
74          }
```

```
75          if (READ_FLAG(PAD2_ON))
76          {
77                  pad2_counter++;
78                  if (pad2_counter == TIMEOUT_PAD_COUNTER)
79                  {
80                          pad2_counter = 0;
81                          CLEAR_FLAG(PAD2_ON);
82                          lcd_pad_state(2,OFF);
83                  }
84          }
85          if (READ_FLAG(PAD3_ON))
86          {
87                  pad3_counter++;
88                  if (pad3_counter == TIMEOUT_PAD_COUNTER)
89                  {
90                          pad3_counter = 0;
91                          CLEAR_FLAG(PAD3_ON);
92                          lcd_pad_state(3,OFF);
93                  }
94  }
95
96          /*********************************************************/
97          /*            Rotinas periodicas dos modulos             */
98          /*********************************************************/
99          lcd_periodic(); //Para controle dos dados do LCD
100
101         GLOBAL_INTERRUPT_ENABLE;
102 }
```

Arquivo “timer1.h”:

```
1  #ifndef          __TIMER1_H_
2  #define          __TIMER1_H_
3
4  /*****************************************************************/
5  /*                       Variaveis globais                       */
6  /*****************************************************************/
7  unsigned char flags_global;
8
9  /*****************************************************************/
10 /*          Prototipos das funcoes publicas do modulo            */
11 /*****************************************************************/
12 void TIMER1_init(void);
```

```
13
14 #endif
```

Arquivo “timer2.c”:

```
1  /*****************************************************************/
2  /*                        Modulo do Timer 2                      */
3  /*****************************************************************/
4
5  //Inclusoes modulos necessarios
6  #include "main.h"
7  #include "timer2.h"
8
9  //Timer para a geracao da saida PWM.
10
11 //A frequencia do PWM esta configurada para ser 62,5kHz, quase 4
       vezes a frequencia
12 //de amostragem do audio gravado no cartao SD.
13
14 //O ideal e que a frequencia do PWM fosse 10 vezes a frequencia
       de amostragem do audio,
15 //porem, para ter resolucao de 8bits no duty cycle, a maxima
       frequencia que pode ser
16 //gerada no PWM e 62,5kHz. Essa limitacao se deve a frequencia de
        oscilacao do sistema,
17 //que e de 16MHz. -> 16MHz/(256bits) = 62,5kHz
18 void TIMER2_init(void){
19
20         // Timer/Counter 2 initialization
21         // Clock source: System Clock
22         // Clock value: 16000,000 kHz
23         // Mode: Fast PWM top=0xFF
24         // OC2A output: Non-Inverted PWM
25         // Timer Period: 0,016 ms
26
27         //Frequencia do PWM -> 62,5kHz e periodo de 16us
28         TCCR2A = (1<<COM2A1) | (1<<WGM21) | (1<<WGM20);
29         TCCR2B = (1<<CS20);
30
31         OCR2A=0x00; //Registrador do duty cycle (8 bits - 0x00 a
               0xFF)
32         //Deve ser carregado a cada interrupcao do timer que esta
33         //sincronizado com a frequencia de amostragem do audio.
34
35         //O pino da saida OC2A (PWM) deve ser configurado como
               saida. (PORTB pino 4 - Arduino pino 10)
```

```
36          DDRB = (1 << DDB4);
37  }
```

Arquivo “timer2.h”:

```
1   #ifndef          __TIMER2_H_
2   #define          __TIMER2_H_
3
4   /*****************************************************************/
5   /*          Prototipos das funcoes publicas do modulo            */
6   /*****************************************************************/
7   void TIMER2_init(void);
8
9   #endif
```

Arquivo “timer3.c”:

```
/*****************************************************************/
/*                        Modulo do Timer 3                      */
/*****************************************************************/

//Inclusoes modulos necessarios
#include "main.h"
#include "timer3.h"

//Timer para gerar interrupcao responsavel por atualizar o valor do duty cycle do PWM.
//O timer esta sincronizado com a frequencia de amostragem do arquivo WAV de 16kHz
void TIMER3_init(void){
        // Timer/Counter 0 initialization
        // Clock source: System Clock
        // Clock value: 2000,000 kHz
        // Mode: Normal top=0xFF
        // Timer Period: 0,0625
        TCCR0B=(0<<WGM02) | (0<<CS02) | (1<<CS01) | (0<<CS00);
        TCNT0=0x83; //Valor para gerar interrupcao a cada 62,5us (16kHz)

        // Timer/Counter 0 Interrupt(s) initialization
        TIMSK0=(0<<OCIE0B) | (0<<OCIE0A) | (1<<TOIE0);
}

//Rotina de interrupcao
ISR(TIMER3_OVF_vect){
        GLOBAL_INTERRUPT_DISABLE;

        // Reinitialize Timer 0 value
        TCNT0=0x83; //16kHz

        //Atualiza valor do duty cycle do PWM com o valor do byte lido do arquivo WAV
        if(sample < (sounddata1_length-1) ){
                OCR2A = pgm_read_byte(&sounddata1_data[sample]);
                sample++;
        }else
        {
                sample++;
```

```
38                  if (sample == (sounddata1_length + 8000) ){
39                          sample=0;
40                  }
41          }
42          GLOBAL_INTERRUPT_ENABLE;
43  }
```

Arquivo “timer3.h”:

```
1  #ifndef          __TIMER3_H_
2  #define          __TIMER3_H_
3
4  /*****************************************************************/
5  /*          Prototipos das funcoes publicas do modulo            */
6  /*****************************************************************/
7  void TIMER3_init(void);
8
9  #endif
```

Arquivo “usart3.c”:

```
1
2  /*****************************************************************/
3  /*                        Modulo da USART3                       */
4  /*****************************************************************/
5  //Este modulo serve tanto para receber dados dos XBee's quanto
       para
6  //enviar dados via MIDI para o computador
7
8  //Inclusoes modulos necessarios
9  #include "main.h"
10 #include "usart3.h"
11 #include "lcd.h"
12
13 //Macro para definir o tamanho do buffer de saida
14 #define BUFFER_LENGHT 256
15
16 //Declaracao e inicializacao de variaveis do modulo
17 unsigned char i=1; //Variavel utilizada no controle do buffer de
       saida
18 unsigned char y=1; //Variavel utilizada no controle do buffer de
       saida
19 unsigned char bufferOUT[BUFFER_LENGHT]; /* Buffer de saida
       */
20 volatile unsigned char dataRX=0;
21
22
23 //Inicia a transmissao pela serial dos caracteres armazenados no
       buffer.
24 void USART3_start_tx (void)
25 {
26         UDR3 = bufferOUT[y];
27         y++;
28 }
29
30 //Adiciona um caractere a ser transmitido no buffer de saida. "x"
       eh caractere a ser transmitido.
31 void USART3_put_char (char x)
32 {
33         bufferOUT[i] = x;
34         i++;
35 }
```

```
36
37 //Limpa o buffer de transmissao.
38 void USART3_clear_buffer_tx (void)
39 {
40         unsigned char k;
41         for(k=1;k < i;k++){
42         bufferOUT[k] = 0;
43 }
44 }
45
46 //Inicializacao do modulo
47 void USART3_init(void){
48
49         // USART3 initialization
50         // Communication Parameters: 8 Data, 1 Stop, No Parity
51         // USART3 Receiver: On
52         // USART3 Transmitter: On
53         // USART3 Mode: Asynchronous
54         // USART3 Baud Rate: 31250
55         UCSR3B= (1<<TXCIE3) | (1<<TXEN3) | (1<<RXEN3) | (1<<
              RXCIE3);
56         UCSR3C= (1<<UCSZ31) | (1<<UCSZ30);
57         UBRR3H=0x00;
58         UBRR3L=0x1F;//31 decimal p/ baud 31250
59
60 //Rotina de interrupcao de transmissao
61 ISR(USART3_TX_vect){
62
63         GLOBAL_INTERRUPT_DISABLE;
64
65         //Verifica se ainda tem dados para enviar
66         if(y < i){
67                 USART3_start_tx();
68         } else{ //Se nao tem mais dados para enviar
69                 USART3_clear_buffer_tx();
70                 i=1;
71                 y=1;
72         }
73
74         GLOBAL_INTERRUPT_ENABLE;
75         return;
76 }
```

```
77
78 //Rotina de interrupcao de recepcao
79 ISR(USART3_RX_vect){
80
81         GLOBAL_INTERRUPT_DISABLE;
82
83         dataRX = UDR3; //Guarda valor recebido na variavel dataRX
84         SET_FLAG(SERIAL_RX); //Liga flag para indicar que recebeu
               dado
85
86         GLOBAL_INTERRUPT_DISABLE;
87         return;
88 }
89
90 /*************************************************************/
91 /*           Rotina para enviar uma mensagem MIDI            */
92 /*************************************************************/
93 void USART3_send_MIDI (unsigned char status, unsigned char note,
      unsigned char velocity){
94         USART3_put_char(status);
95         USART3_put_char(note);
96         USART3_put_char(velocity);
97         USART3_start_tx();
98 }
```

Arquivo “usart3.h”:

```
1
2  #ifndef          __USART3_H_
3  #define          __USART3_H_
4
5  /*****************************************************************/
6  /*          Prototipos das funcoes publicas do modulo            */
7  /*****************************************************************/
8  void USART3_init (void);
9  void USART3_clear_buffer_tx (void);
10 void USART3_start_tx (void);
11 void USART3_put_char (char x);
12 void USART3_send_MIDI (unsigned char status, unsigned char note,
      unsigned char velocity);
13
14 #endif
```

Arquivo “WiDrum.ino”:

```
#include <SPI.h>
#include <SD.h>

File myFile;

//Inclusoes dos "Headers" dos modulos
#include "main.h"
#include "usart3.h"
#include "timer1.h"
#include "timer2.h"
#include "timer3.h"

/*****************************************************************/
/*              Macros para uso na funcao loop                   */
/*****************************************************************/
#define ID_PAD    0xD0    //Byte para identificar se o dado
   recebido eh de um sensor
#define PAD1    ID_PAD+1  //ID do PAD 1
#define PAD2    ID_PAD+2  //ID do PAD 2
#define PAD3    ID_PAD+3  //ID do PAD 3

/*****************************************************************/
/*                Macros para comunicacao MIDI                   */
/*****************************************************************/
//Status Bytes
#define NOTE_ON   144
#define NOTE_OFF  128

//Data Bytes
#define BASSDRUM  36
#define SNARE   40
#define HIHAT   46
#define RIDE    51
#define CRASH   49
#define TOM1    47
#define TOM2    45
#define TOM3    43

volatile unsigned char flags_global;
unsigned char flagSensorOK=0;

```

```
41 void setup() {
42
43         if (!SD.begin(53)) {
44                 return;
45         }
46
47         //Inicializacao dos modulos
48         USART3_init();
49         TIMER1_init();
50         TIMER2_init();
51         TIMER3_init();
52
53         GLOBAL_INTERRUPT_ENABLE; //Habilita interrupcao global
54 }
55
56 void loop() {
57
58
59 /*************************************************************/
60 /*Rotina para verificar e tratar dado recebido pela serial  */
61 /*************************************************************/
62
63         if (READ_FLAG(SERIAL_RX)){
64                 CLEAR_FLAG(SERIAL_RX);
65
66                 if ( (flagSensorOK == 0) && ((dataRX & 0xF0) ==
                       ID_PAD) ){
67                         flagSensorOK = dataRX & 0x0F;
68                 } else {
69                         switch (flagSensorOK){
70                                 case 1:
71                                         USART3_send_MIDI(NOTE_ON,
                                           BASSDRUM,dataRX);
72                                         SET_FLAG(PAD1_ON);
73                                 break;
74
75                                 case 2:
76                                         USART3_send_MIDI(NOTE_ON,
                                           SNARE,dataRX);
77                                         SET_FLAG(PAD2_ON);
78                                 break;
79
```

```
80                                  case 3:
81                                          USART3_send_MIDI(NOTE_ON,
                                               HIHAT,dataRX);
82                                          SET_FLAG(PAD3_ON);
83                                  break;
84                          }
85                          flagSensorOK = 0;
86                  }
87          }
88  }
```

Arquivo “main.h”:

```
1   #ifndef          __MAIN_H
2   #define          __MAIN_H_
3
4   #include <avr/io.h> //Para acesso as portas de entrada/saida
5   #include <avr/interrupt.h> //Para uso de interrupcoes
6   #include <avr/pgmspace.h> //Para acesso aos dados armazenados na
       memoria de programa (flash)
7
8   /*****************************************************************/
9   /*                    Variaveis globais                          */
10  /*****************************************************************/
11  extern volatile unsigned char dataRX; //Guarda o valor recebido
       pela serial
12  extern volatile unsigned char flags_global; //Guarda as flags
13  extern volatile uint16_t sample;
14
15  /*****************************************************************/
16  /*          Bits (flags) da variavel "flags_global"              */
17  /*****************************************************************/
18  //Utilizados para "comunicacao" entre ISR e laco principal.
19
20  #define SERIAL_RX                               (1 << 0) //Indica
       recebimento pela serial
21  #define STARTUP_SCREEN_DONE             (1 << 1)
22  #define KEY_ENCODER                             (1 << 2)
23  #define MENU                                    (1 << 3)
24  #define PAD1_ON                                 (1 << 4)
25  #define PAD2_ON                                 (1 << 5)
26  #define PAD3_ON                                 (1 << 6)
27
```

```
28 /*****************************************************************/
29 /*      Macros para facilitar a leitura e controle das flags     */
30 /*****************************************************************/
31 #define SET_FLAG(BYTE) flags_global |= BYTE
32 #define CLEAR_FLAG(BYTE) flags_global &= ~(BYTE)
33 #define READ_FLAG(BYTE) ((flags_global & BYTE) == BYTE) ? (1) :
      (0)
34 #define OFF                         0
35 #define ON                          1
36
37 /*****************************************************************/
38 /*      Macros para habilitar e desabilitar interrupcoes         */
39 /*****************************************************************/
40
41 #define GLOBAL_INTERRUPT_ENABLE          sei()
42 #define GLOBAL_INTERRUPT_DISABLE         cli()
43
44 #endif
```